JIO
JBUS to PCI Bridge ASIC

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
1 800 555-9SUN

Part No.: SME2530PBGA
Last Technical Updated -- 7/20/07

Please
Recycle

Contents

CHAPTER 1

JIO Basics

1.1 Overview

JIO is a companion core-logic ASIC to the **Sparc III** *i*-series 64-bit SPARC V9 CPU. JIO and Sparc IIIi communicate through a split transaction 16-byte shared address/data bus, which is known as Jbus. The central task of JIO is to be the point of access to I/O, graphics, and system interrupts, for a uni-processor up to a four-way symmetric multi-processor Sparc IIIi-based system. For those familiar with the Intel system hardware partitioning, JIO is the equivalent of an Intel Northbridge, minus the interface to main-memory. The interface to main-memory comes directly off the Sparc IIIi CPU in our case. It is allowable for a given system to be configured with more than one JIO, if added I/O bandwidth is desired.

The main system interfaces JIO offers are:

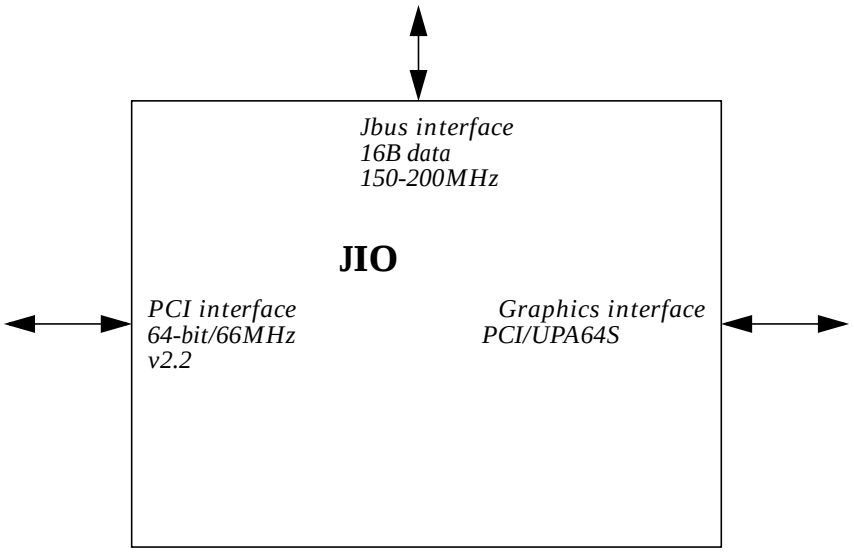
- Jbus
- 66MHz/64-bit or 33 Mhz/32 bit v2.2 Peripheral Component Interconnect (PCI), capable of supporting up-to eight external masters w/internal arbiter.
- Multiplexed graphics, which can be configured to be one of: UPA64S, or 66MHz/64-bit or 33 Mhz/32 bit v2.2 PCI (four external masters).

1.2 Component Overview

The JIO ASIC contains these major logical components:

- Jbus Interface Unit
- PCI Interface Unit
- Graphics Interface Unit
- PCI I/O Cache Unit
- Graphics I/O Cache Unit
- Synchronizer Unit

FIGURE 1-1 JIO Interface Overview



Interface Overview

1.2.1 Jbus Interface Unit

The Jbus unit detects which transactions present on Jbus are targeted for JIO, accept/queue those transactions, and coordinate with the appropriate destination unit within JIO to which the address and data are sent. When one of the CPU

modules is the target, the converse is true. The Jbus unit is responsible to send out onto Jbus transactions initiated from within JIO (such as from the PCI unit, Graphics unit, I/O caches, etc.). Major blocks are defined in Table 1-1.

TABLE 1-1 Jbus Unit Major Blocks

Name	Description
Jbus Arbitration unit	Arbitrates for Jbus on a cycle-by-cycle basis between the external agent(s) attached to Jbus and the two out-going request streams from within JIO (PCI leaf and Graphics leaf).
Transaction encoder	Encode outgoing transactions from JIO -> Jbus
Transaction decoder	Decode incoming transactions from Jbus -> JIO
Parity generator	Generates parity for outgoing transactions
Parity detector	Checks parity for incoming transactions
Input/output register bank	Both incoming and outgoing signals coming from/to Jbus are registered.
Flow control unit	Looks at global AOK state in the system as well as DOK state for all ports in the system.
Interrupt ACK/NACK unit	Sends interrupt ACK/NACK information, and interrupt target ID to JIO interrupt controller (located in PCI unit).
CSR bank	Jbus CSR's

1.2.2 PCI Interface Unit

The standalone PCI interface on JIO is a 64/32-bit, 33/66MHz version 2.2 compliant implementation. The interface has an internal arbiter that can support up-to eight masters resident on the bus. Support for external arbitration is also present. It is intended for this leaf interface to offer very similar functionality as the PCI_A bus on the SUPCI ASIC(Sun Safari Bus to UPA and PCI Bridge ASIC)SUPCI, and also be as close as possible from a software view of this leaf interface to that of PCI_A on SUPCI. One point to keep in mind is this interface on JIO is PCI v2.2 compliant, and SUPCI's PCI_A bus is v2.1 compliant.

Hardware wise, one main difference is that JIO does not have the streaming cache unit that SUPCI uses in this interface. JIO uses the "PCI I/O Cache" unit described in an up-coming section, which is not a streaming cache.

The PCI interface unit includes an interrupt block that receives system interrupts from the external I-chip, and also deals with interrupts generated from activity within the PCI interface unit itself. The interrupt unit communicates with the Jbus cluster to send out the interrupt onto Jbus.

1.2.3 Graphics Interface Unit

The graphics interface supports two different protocols, only one of which may be in use at a given time. UPA64S, and 64/32-bit, 33/66MHz (v2.2) PCI are supported. JIO's graphics interface would be set up in UPA64S or PCI mode by strapping its *PAD_G_UPA_PCI_SEL* pin to 1 or 0 respectively in the system.

As with the standalone PCI bus interface unit, the graphics interface unit interfaces with an I/O cache, when configured in the PCI mode. The I/O caches on JIO are basically instances of the same design, and differ in very minor ways.

1.2.3.1 PCI Mode

The PCI protocol mode has an internal arbiter that supports eight masters on the PCI bus. Currently only four external request lines are supported though for this PCI interface. An external arbiter is also supported. This is a fully functional standalone PCI bus. Please refer to the PCI spec. With the exception of support for external arbiter support and spec version 2.2 compliance, the functionality and software view of this interface has been made as compatible with SUPCI as possible.

1.2.3.2 UPA64S Mode

UPA64S is a slave only implementation of UPA that is used to attach to Sun proprietary graphics accelerators. The logic to implement this interface essentially converts PIO transactions received over Jbus and generate PIO transactions on the UPA64S bus. Please refer to the UPA spec and the UPA64S implementation on SUPCI for further information. The functionality of the UPA64S implementation on JIO has been made as software compatible to that on SUPCI as possible. The UPA64S implementation has been done to support 200 MHz operation of the UPA interface, but this is an implementation detail that should have little or no impact on software.

1.2.4 PCI I/O Cache Unit

The PCI I/O cache has 8 entries, each of which contains 64 bytes of data. The purpose of the cache is to improve DMA performance. The I/O cache also has the ability to perform data prefetching, to aid in minimizing DMA read latency (JIO is returning data to the esternal PCI master). This prefetching can be disabled, if so desired, by software (via a CSR).

1.2.5 Graphics I/O Cache Unit

See above description of the PCI I/O cache. The I/O cache located in the graphics cluster is very similar to that located in the standalone PCI interface.

1.2.6 Synchronizer Unit

All synchronizers used in JIO will reside in the top-level synchronizer unit.

CHAPTER 2

Differences From SUPCI

The software view of JIO differs from SUPCI in several areas, including:

- Functionality Differences
- Control/Status Registers (CSRs)
- Address Space Size
- System Interface

This chapter summarizes these differences.

2.1 Functionality Differences

The following summarizes areas of functional difference between JIO and SUPCI.

- SUPCI has 5 bit Node ID and 5 bit agent ID, whereas in the Sparc IIIi-JIO system, each Jbus agent is specified by a 5 bit JPID, where JPID[3:0] are hardwired from external pins and JPID[4] is programmable by software.
- JIO implements fully set associative Writeback I/O caches, while SUPCI implements streaming caches. A new set of registers are being implemented to support these I/O caches.
- PCI Leaf in JIO does not support diagnostic loop-back mode.
- PCI leaf in SUPCI supports dynamic change of clock frequency based on bus activity level in Energy Star (E*) mode. JIO does not support such dynamic clock frequency changes.
- JIO does not support detection and logging of PCI streaming byte hole errors as it does not support streaming and consistent modes.
- I/O Cache Flush to Memory:

  There is no way for OS to flush I/O cache in normal operation unless we know what addresses might be in the I/O cache. In case we do not know the address, only way to do that would be is to first evict all dirty lines in the I/O cache by

doing a bunch of DMA reads (displacement flush). Then diagnostic PIO stores can be used to invalidate clean lines. While the PIO stores are occurring, any new DMA may cause the clean lines to go dirty again. Hence the stores have to be issued only when no I/O DMA activity is in progress.

Alternatively if the address is known, CPU can execute a CAS that does not swap. It will get dirty copy from the I/O cache into the CPU cache. Note that here the flush does not go to memory directly, but the CPU can later evict it to memory.

Note: The I/O cache is always as coherent as any of the CPU caches. So the expectation will be that Solaris would not need to flush the I/O cache during normal operation.

- There is no way to transmit ECC syndrome information over Jbus in Sparc IIIi-JIO based system. So software has to handle this by getting the syndrome from the Memory Controller.
- JIO IOMMU logs translation errors in IOMMU Control Register in addition to logging them in the IOMMU CAM itself (like is Sabre) to give software more visibility into error state and type in IOMMU. This feature is not present in SUPCI.
- JIO differs from SUPCI in certain ways in the areas of PCI Address and Data Parity detection and reporting mechanisms. The differences are :
  - On detection of PCI address parity error, SUPCI does not assert SERR#, while JIO does assert SERR# if SERR_EN = PER = 1.
  - SUPCI sets SSE when it detects PCI address parity error or if it sees SERR# assertion by another PCI device provided SERR_EN = PER = 1. JIO sets SSE if and only if it detects PCI address parity error provided SERR_EN = PER = 1.
  - SUPCI sets PCI_SERR if SERR# gets asserted by any PCI device and SERR_EN = 1. JIO sets PCI_SERR if and only if SERR# is asserted by any PCI device.
  - SUPCI issues target abort on detecting PCI address parity error if PER =1 and the PCI address appears to be a valid DMA address that SUPCI should normally respond to for a SAC or DAC-2nd cycle . For DAC-1st cycle, SUPCI would assert target abort on detecting PCI address parity error on any PCI address if PER = 1. JIO asserts target abort on detecting PCI address parity error on any PCI address if SERR_EN = PER = 1.
  - SUPCI relies on external master to issue interrupt on a DMA write data parity error with SUPCI as a target , and does not assert interrupt itself. JIO however lets the bad data go through to memory on a DMA write data parity error with JIO as a target. Hence it cannot rely on external master for the interrupt, and generates an interrupt itself.

2.2 Control/Status Registers (CSRs)

The main area of CSR differences are in the Jbus interface. JIO is essentially replacing Safari with Jbus.The same registers are used as in SUPCI, but the bit allocations have changed to support Jbus as opposed to Safari. Also the Interrupt Packet Format on Jbus is different from that on Safari Bus in SUPCI. The NewLink related registers are unimplemented in JIO.SUPCI implements a Queue control Register which is unimplemented in JIO.

- A new set of registers are used for the 2 I/O caches, the streaming registers in SUPCI are unimplemented in JIO. All CSR bits related to Streaming in IOMMU and PCI blocks are assigned to reserved.
- JIO implements a scratch pad register which is used by OBP to perform some calculations prior to having memory configured in the MP system.
- Each PCI leaf module in JIO will implement a set of registers (PCI target retry limit, PCI Target Latency Timer) to avoid deadlock cases. These deadlock cases are described in detail in Chapter 5 in the Jbus System Spec.
- JIO has a new set of registers (Reset_Gen, Reset_Source) to support Reset.
- PCI Consistent DMA Flush/Sync Register not implemented. Software DMA flush is handled differently in JIO through Ichip PCI DMA Flush/Sync Pending and Complete Registers. In SUPCI, software would write the PCI Consistent DMA Flush/Sync Register register with a physical address which would then cause the flush.
- JIO does not implement IOMMU LRU Queue Diag Registers. The IOMMU in JIO is derived from Sabre. The Sabre IOMMU does not have a LRU queue.
- JIO does not implement the PCI Idle Check Diag Register. SUPCI uses this register to monitor the idle/busy status of a set of blocks like MMU, MDU, STC etc. A general event monitor would be used to monitor the idle/busy status of the different blocks.

2.3 Address Space Size

UltraSPARC-III extends both the virtual and physical address space implemented in UltraSPARC-I. UltraSPARC-III implements the full 64 bit virtual address range defined in the SPARC-V9 architecture (No VA hole as in UltraSPARC-I). The physical address range has also been extended from 41 bits to 43 bits.

As on the Jalapeño CPU, the width of a physical address on JIO is defined to be 43-bits. This physical address space can be accessed through virtual-to-physical address mapping by the PCI interface IOMMU, the graphics interface IOMMU, or by invoking the bypass mode on the IOMMUs.

Via the Jbus specification, all Jbus agents (devices attached to Jbus) have a 64 Gbyte cacheable memory space and a 64 Gbyte non-cacheable memory space assigned to them.In addition they have an 8 Mb of non-cacheable config space where the internal registers reside.

Thus JIO would respond to a split address mapping in the noncacheable space. The first segment is a 8 Mbyte section where most of JIO's internal registers are mapped. The second segment is a 64 Gbyte of space allocated to PCI config/IO/Mem and UPA. Please refer to Chapter 3 "Address Spaces and Translations" for a detailed description for the address format for these regions.

2.4 System Interface

- Jbus replaces the Safari bus interface that SUPCI uses.
- PCI bus controllers in JIO are v2.2 compliant, while those in SUPCI are v2.1 compliant.
- JIO does not implement NewLink but has a PCI/UPA64S multiplexed graphics port.
- JIO would support up to 8 bus masters on the Primary PCI bus, SUPCI supports up to 6.

2.5 Summary of CSR Differences with SUPCI

2.5.1 Registers Changed from SUPCI

- Safari Device ID Register bits 26:22 reserved, return zeroes on read
- "Address Match registers" renamed to "Offset Base Registers"
- "Address Mask registers" renamed to "Offset Mask Registers"
- In Offset Match Register, bits 42:36 reserved, return zeroes on reads

- In Offset Match register, “match” field renamed to “base”.
- In SUPCI Control/Status Register, bits 14:0 are reserved, return zeroes on read
- In SUPCI Control/Status Register, bit 24 is R/W and resets to 1’b1.
- In SUPCI Control/Status Register, bits 26:25 are Reserved.
- In SUPCI Control/Status Register, bits 28:27are R/W and indicate “Arbitration Mode”
- In SUPCI Control/Status Register, bit 29 is R/W and indicates “Control Parity Gen/Det Mode”
- In SUPCI Control/Status Register, bits 35:34 are read only and indicate “DTL Mode”
- In SUPCI Control/Status Register, bits 42:36 are R/W and indicate “Jpack_Delay”
- In SUPCI Control/Status Register, bit 43 is R/W and indicate “Control Parity Error Detection Enable”.
- In SUPCI Control/Status Register, bit 44 is R/W and indicates “Par_Delay”.
- In SUPCI Control/Status Register, bits 46:45 indicate L5CLK_Sel bits to select one out of all internal JIO Clocks for Jitter management.
- In SUPCI Control/Status Register, bits 52:47 indicate PLL tune bits[5:0] for Jbus PLL.
- In SUPCI Control/Status Register, bit 53 indicates “Pulse_JERR” to enable software to send a one clock wide pulse on J_ERR out of JIO for triggering analyzer/scope.
- In SUPCI Control/Status Register, bits 57:54 are reserved.
- Safari Common error bit assignments have changed. Bits [61:58],9,7,0 reserved. The following are the new error assignments for the error bit assignments that have changed:

bit 21 : Snoop Error has happened due to Secondary PCI DMA.

bit 20 : Snoop Error has happened due to Primary PCI DMA.

bit 19 : Foreign RD hitting cache line in S , O or M.

bit 18 : Snoop error due to own WRI hitting cache line in S , O or M.

bit 17 : Snoop error due to own RDS hitting cache line in S , O or M.

bit 16 : Snoop error due to own RDSA hitting cache line in S , O or M.

bit 15 : Snoop error due to own OWN hitting cache line in S or M.

bit 14 : Snoop error due to own RDO hitting cache line in O or M.

bit 13 : Jbus Write Data Cycle parity error

bit 12 : Control Parity Error

bit 11 : Snoop Error

bit 10 : Jbus Illegal Byte Enable Error

bit 8 : Jbus Illegal coherency state install error

bit 6 : Jbus Read Data Cycle parity error

bit 3 : Jbus Timer Count Expiration error

- ECC Control Register renamed to Jbus Parity Control Register
- In Jbus Parity Control Register, bit 18 changed to FCPE (Force Control Parity Error). Bit [17:14] are reserved, return zeroes on reads. Bits[13:10] changed to FDPE (Force Data Parity Error), one bit per 32 bit word. Bits [9:4] are reserved.
- In UE/CE Asynchronous Fault Status Registers, bit 63, bits[61:60],bits[57:56], bits[41:32], bits [23:22], bits [19:13], bits [8:0] are reserved and return zeroes on read.
- In UE/CE Asynchronous Fault Address register, bit 43 is reserved and returns zero on a read.
- In DTag Diagnostic register and CTag Diagnostic register, Bit 63, bit 59, bit 42 are reserved and return zeroes on reads
- In DTag Diagnostic register and CTag Diagnostic register, bits [58:56] reflect the MOESI State of the Cache line in the I/O cache.
- DTag and CTag Diagnostic registers in JIO are writable by software. In SUPCI, they were read only.
- Jbus interface related performance Events to monitor have been changed from Safari Interface. Please look at Table 4-21 for the new set of events to monitor performance of.
- UPA Slot 0/Slot 1 Configuration register bits [32:30] are newly added. These bits need to be programmed to support multiple pending reads from the UPA Graphics Card.
- UPA Slot 0/Slot 1 Configuration register bits [37:33] are newly added. These bits need to be programmed to support multiple pending writes to the UPA Graphics Card.
- UPA Slot 0/Slot 1 Configuration register bits [42:38] are newly added. These bits need to be programmed to support multiple pending transactions to the UPA Graphics Card.
- UPA Slot 0/Slot 1 Configuration register bit 14 is reserved. Reset to zero.
- UPA64S Interface Control Register bits 10:5 are newly added. These bits represent PLL tune bits for Graphics PLL when JIO's Graphic leaf is configured in the UPA mode.
- UPA64S Interface Control Register bit 4 newly added. This bit when set to 1, inhibits JIO's UPA logic from sending stores to an UPA64 device while there is an outstanding read to an UPA64S device.

- UPA64S Interface Control Register bits 2 and 3 are newly added. These bits should be programmed by software to control the sample point of UPA read data properly w.r.t to the skewed UPA read clock.
- UPA64S Interface Control Register bits [1:0] are reserved.
- PCI Control & Status Registers bits 63,51,[50:48],35,33,18,8 are reserved and return zeroes on reads
- ARB_EN field in PCI Control & Status Register is expanded to 8 bits (bits [7:0]) as JIO would support a maximum of 8 external PCI bus masters.
- PCI Control & Status Register bits 14:9 represent PLL tune bits for the PLL associated with the corresponding leaf (PCIA / PCIB).
- PCI Control & Status Register bit 16 (ARB_PARK bit) reset state has been changed to a 1 (previous bus owner parked on PCI bus).
- PCI Control & Status Register bit 23:21 is changed to TRGT_RW_STL_WT.
- PCI Control & Status Register bit 26 is changed to FRC_TRGT_RTRY.
- PCI Control & Status Register bit 27 is changed to FRC_TRGT_ABRT.
- PCI Control & Status Register bit 28 changed to PEN_RD_LINE.
- PCI Control & Status Register bit 29 changed to PEN_RD_ONE.
- PCI Control & Status Register bit 30 changed to PEN_RD_MLTPL.
- PCI Control & Status Register bit 60:52 is changed to ARB_PRIORITY[8:0]
- PCI Control & Status Register bit 61 changed to DTO_INT_EN
- PCI Control & Status Register bit 62 changed to PCI_DTO_ERR
- In PCI AFSR Register, bits 58,52,[41:40] are reserved and return zeroes on reads
- In PCI AFSR Register, bits [39:32] represent PCI Byte Masks [7:0] for transfer in Error
- In PCI Diagnostic Register, bits 7,10 are reserved.
- IOMMU Control Register bit 24 represents ERR, bits [26:25] represent ERRSTS
- IOMMU TLB Tag Diagnostics Access Register bit 20 is reserved. Would return zero on a read
- In Format of Interrupt Mapping Register (Table 4-61), bits [25:21] are reserved, and return zeroes on reads.
- PCI Performance events to monitor have been changed. Please look at table 4-70 for the new set of events to monitor performance of.

2.5.2 Registers Dropped from SUPCI

- NewLink Address Match Register not implemented
- NewLink Address Mask Register not implemented

- NewLinkAlt Address Match Register not implemented
- NewLinkAlt Address Mask Register not implemented
- SUPCI Queue Control Register not implemented
- Safari Debug Registers not implemented
- PCI Energy Star Register not implemented
- IOMMU LRU Queue Diagnostic Registers not implemented
- Streaming Cache related registers not implemented
- PCI Idle Check Diag Registers are dropped
- UPA Energy Star Control Register is dropped.
- PCI Consistent DMA Flush/Sync Register not implemented. Software DMA flush is handled differently through Ichip PCI DMA Flush/Sync Pending and Complete Registers.

2.5.3 Registers Added to SUPCI

- Reset_Gen Register added (refer to sec 7.1.3.1)
- Reset_Source Register added (refer to sec 7.1.3.2)
- UPA64S Reset Control Register added (refer to sec 4.2.13)
- Jbus Change Initiation Control Register added (refer to sec 4.2.9)
- General-Purpose Input/Output (GPIO) registers added (refer to sec 4.2.14)
- Scratch Pad Register Added (refer to sec 4.4)
- PCI Target Retry Limit Register Added (refer to sec 4.5.1.5)
- PCI Target Latency Timer Register Added (refer to sec 4.5.1.6)
- PCI Target Space Register added (refer to sec 4.5.1.7). This register is visible both from CPU and PCI sides.
- PCI Target Error VA Log Register added (refer to sec 4.5.1.8)
- PCI Extension Register added (refer to sec 4.5.1.9)
- 3 new I/O cache registers added per PCI leaf for JIO (refer to sec 4.5.2)
- Translation Fault Address Register added in IOMMU (refer to sec 4.5.3.6)
- Interrupt Routing Register added in PCI A Leaf of JIO. This register would be initialized by OBP to route the interrupts into PCI-A bus or PCI-B bus. This register is invisible to Unix OS (refer to sec 4.5.1.6).
- JIO implements a new device node representing the Ichip to re-architect the interrupt scheme in to a more manageable layout. Since the current Interrupt registers are all located in the PCI CSR address space, all of those registers are aliased to this new Ichip register bank that represents the Ichip nexus.The only exception is that the PCI Consistent DMA Flush/Sync Registers for the 2 PCI

domains are reserved and a new set of registers called DMA Flush/Sync Pending and Complete registers are used by software to synchronize on DMA writes on the two PCI buses (refer to sec 4.5.7).

CHAPTER 3

Address Spaces and Translations

This chapter documents the regions of physical address space which JIO can generate transactions for, or will respond to on each of its interfaces (Jbus, PCI, and graphics). Recall that the graphics interface can be one of: PCI, UPA64S. This chapter also documents the modifications made to addresses as they pass from one interface to another. The following areas are covered:

- Physical Address Space
- Jbus => Leaf Interfaces
- UPA => Jbus
- PCI => Jbus

All JIO operations involve at least one transaction on Jbus. Transactions which logically transfer data between two leaf interfaces (such as PCI and graphics) do so by way of Jbus, so they can conceptually be broken up into separate Leaf⇒Jbus and Jbus⇒Leaf transactions which are documented below. Transactions in which JIO does not participate (e.g. peer-to-peer transfers between two devices on the same PCI bus for example) are not documented here.

3.1 Physical Address Space

As on the Jalapeño CPU, the width of a physical address on JIO is defined to be 43-bits. This physical address space can be accessed through virtual-to-physical address mapping by the PCI interface IOMMU, the Graphics interface IOMMU, or by invoking the bypass mode on the IOMMUs.

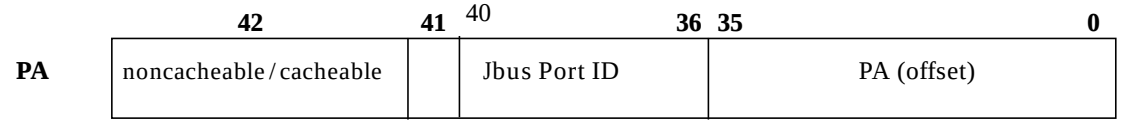
Via the Jbus specification, all Jbus agents (devices attached to Jbus) have a 64 Gbyte cacheable space and a 64 Gbyte noncacheable space assigned to them. JIO will utilize only the noncacheable space, since it does not have an interface to main-memory. Although, JIO is still responsible to respond to reads/writes to it's cacheable space to avoid system hangs.For reads to its cacheable space, JIO returns Read Error(error type: Unmapped Error). Writes to its cacheable space are simply ignored (they

simply disappear on Jbus), but are logged as Unmapped Error in Jbus Error Log Register.Moreover, accesses to JIO's NC space outside the regions mapped by the Address Mask / Match registers are treated as Unmapped Error.

Under the current Jbus protocol, up-to 8 Jbus agents are supported. This number may be as high as 64 theoretically, if a Jbus<->Jbus bridge was designed / employed in a Jbus-based system.

JIO would respond to a split address mapping in the noncacheable space. The first segment is a 8 Mbyte section where most of JIO's internal registers are mapped. The second segment is a 64 Gbyte of space allocated to PCI config / IO / Mem and UPA. Even though JIO may not require the use of the entire 64 Gbyte of noncacheable address space, it must respond to requests received over Jbus that may lie within a region or regions of noncacheable space that is unused. It would respond with undefined data.

FIGURE 3-1 Physical Address Formation in a Jbus-based System for the 64GB address space



noncacheable space : PA[42] = 1
cacheable space : PA[42] = 0
for cacheable memory space : PA[41] = X
for noncacheable slave memory/IO/Config space : PA[41] = 1

The reason for this is to help prevent system hangs, as in the case with the unused cacheable space on JIO described above. So, in a nutshell, JIO owns 128 Gbyte total of slave address space, and 8 MB of Config Space and is responsible for responding to all types of requests to access that space.

JIO's cacheable address range is 64 Gbytes of memory starting from the Base Address defined as **JIO_CBase** = {0,0,JPID,(36'b0)}. JIO's noncacheable address range is also 64 Gbytes of memory starting from the Base Address defined as **JIO_NCBase** = {1,1,JPID[4:0], (36'b0)}.

Note: Although JIO hosts 2 Jbus ports (PCI and Gr ports), they share one 64 Gbyte noncacheable memory space, one 64 Gbyte cacheable memory space, and one 8 MB config space. So in any configuration, the memory space and config space associated with JIO's J_ID +1 is aliased to the address space of J_ID.

3.2 Jbus => Leaf Interfaces

3.2.1 Jbus Cacheable Space

JIO is not the final repository for any address in cacheable domain. JIO does have two I/O caches (one for PCI interface and one for the graphics interface), so it does respond to cacheable transactions by returning dirty data when necessary.However, JIO has a 64 Gbyte of unused cacheable space. It is responsible for returning undefined data for cacheable reads issued by any Jbus Master to this space, and ignoring any writes to the same space.

3.2.2 Jbus Noncacheable Space

JIO decodes and responds to up to 6 separate regions in Jbus noncacheable space. The first region is an 8Mb region that is allocated to every Jbus agent. It is located at physical address **Jbus_Base**. In addition, there are two programmable regions each for PCI-A, PCI-B and one for UPA which are mapped via offset mask and base registers in the Jbus Interface Block.These programmable regions are based off of **JIO_NC_Base**. The regions are summarized in Table 3-1. The Base Address IDs from the table are used later to refer to these regions. BE/LE indicates whether the regions use a big-endian convention or little-endian convention for byte ordering.

Table 3-1 JIO Decoding of Jbus Noncacheable Space

Base Address ID	Usage	BE/ LE	Size	Supported Jbus Transactions	Generated Leaf Transaction
Jbus_Base = {1,0,13'b0, JPID[4:0], 23'b0}	Maps internal registers for all JIO Leaf Blocks	BE	8 MB	NCRD NCWR	Internal register access
Upa0Base (programmed in Offset_Base register)	Maps UPA64S Slot 0 slave space	BE	2-8 GB (as per bits 35:24 in Offset_Mask register)	NCRD NCBRD NCWR NCBWR	UPA P_NCRD_REQ UPA P_NCBRD_REQ UPA P_NCWR_REQ UPA P_NCBWR_REQ
PCI-A_MemBase (programmed in Offset_Base register)	Maps PCI-A Memory Space	LE	16MB - 4 GB (as per bits 35:24 in Offset_Mask register)	NCRD NCBRD NCWR NCBWR	PCI memory read PCI Memory read line PCI Memory write PCI memory write line

Table 3-1 JIO Decoding of Jbus Noncacheable Space

Base Address ID	Usage	BE/ LE	Size	Supported Jbus Transactions	Generated Leaf Transaction
PCI-A_CfgIOBase (programmed in Offset_Base register)	Maps PCI-A Configuration Space and PCI-A I/O Space	LE	32 Mb (as per bits 35:24 in Offset_Mask register)	NCRD NCBRD	PCI Config/IO read PCI Config/IO read line
				NCWR NCBWR	PCI Config/IO write PCI Config/IO write line
PCI-B_MemBase (programmed in Offset_Base register)	Maps PCI-B Memory Space	LE	16MB - 4 GB (as per bits 35:24 in Offset_Mask register)	NCRD NCBRD	PCI memory read PCI Memory read line
				NCWR NCBWR	PCI Memory write PCI memory write line
PCI-B_CfgIOBase (programmed in Offset_Base register)	Maps PCI-B Configuration Space and PCI-B I/O Space	LE	32Mb (as per bits 35:24 in Offset_Mask register)	NCRD NCBRD	PCI Config/IO read PCI Config/IO read line
				NCWR NCBWR	PCI Config/IO write PCI Config/IO write line

Some of these regions are further divided into distinct sections with specific meanings (for example, Jbus_Base is divided into multiple sections so that each leaf block has space for mapping its internal registers). These subdivisions are summarized in the following table. Again, the section IDs from the table will be used later to refer to the particular sections within JIO's address regions.

TABLE 3-2 Subdivisions of JIO's Address Regions

Section ID	Usage	BE/ LE	Base Address	Size	Supported Jbus Transaction	Generated Leaf Transaction
Fcode	Jbus device ID register (JIO does not support optional Fcode PROM)	BE	Jbus_Base+ 0x000000	4 Mb	NCRD (8 byte, aligned)	Internal register access
					NCWR (8 byte, aligned)	
JbusCSRBase	Jbus and UPA interface blocks internal registers	BE	Jbus_Base+ 0x400000	1 Mb	NCRD (8 byte, aligned)	Internal register access
					NCWR (8 byte, aligned)	

TABLE 3-2 Subdivisions of JIO's Address Regions

Section ID	Usage	BE/ LE	Base Address	Size	Supported Jbus Transaction	Generated Leaf Transaction
NewLinkCSRBase	Not used in JIO. Writes to these registers are ignored. Reads return "Error" .	BE	Jbus_Base+ 0x500000	1 Mb	NCRD (8 byte, aligned) NCWR (8 byte, aligned)	None
PCI-A_CSRBase	PCI-A leaf internal registers, I/O Cache, IOMMU, Interrupt registers, Streaming cache registers	BE	Jbus_Base+ 0x600000	1 Mb	NCRD (8 byte, aligned) NCWR (8 byte, aligned)	Internal register access
PCI-B_CSRBase	PCI-B leaf internal registers, I/O Cache, IOMMU, Interrupt registers, Streaming cache regsiters	BE	Jbus_Base+ 0x700000	0.5 Mb	NCRD (8 byte, aligned) NCWR (8 byte, aligned)	Internal register access
Ichip-CSRBase	Ichip nexus CSR's. Interrupt related registers in PCI-A and PCI-B CSR banks are aliased to these registers.	BE	Jbus_Base+ 0x780000	0.5 Mb	NCRD (8 byte, aligned) NCWR (8 byte, aligned)	Internal register access
PCI-A_ConfigBase	Maps PCI-A Configuration Space	LE	PCI-A_CfgIOBase + 0x0000000	16 Mb	NCRD (0-4 bytes, spanning single word only) NCWR (0-4 bytes, spanning single word only)	PCI Configuration Read PCI Configuration Write
PCI-A_IOBase	Maps PCI-A I/O Space	LE	PCI-A_CfgIOBase + 0x1000000	16 Mb	NCRD (0-4 bytes, spanning single word only) NCWR (0-4 bytes, spanning single word only)	PCI I/O Read PCI I/O Write

TABLE 3-2 Subdivisions of JIO's Address Regions

Section ID	Usage	BE/ LE	Base Address	Size	Supported Jbus Transaction	Generated Leaf Transaction
PCI-B_ConfigBase	Maps PCI-B Configuration Space	LE	PCI-B_CfgIOBase+ 0x0000000	16 Mb	NCRD (0-4 bytes, spanning single word only)	PCI Configuration Read
					NCWR (0-4 bytes, spanning single word only)	PCI Configuration Write
PCI-B_IOBase	Maps PCI-B I/O Space	LE	PCI-B_CfgIOBase+ 0x1000000	16 Mb	NCRD (0-4 bytes, spanning single word only)	PCI I/O Read
					NCWR (0-4 bytes, spanning single word only)	PCI I/O Write

The Jbus physical address for any given Jbus noncacheable transaction to which JIO will respond can thus be divided as BaseAddress + Offset, where BaseAddress is one of the above regions or sections, and Offset is less than the size of the corresponding region/section. As the physical address is passed to the leaf block, it typically undergoes some transformation, which is documented below for each different destination.

3.2.2.1 Access to UPA

- UPA<38:33> = 000000
- UPA<32:4> = Offset<32:4>

3.2.2.2 Access to PCI

PCI Configuration Space

When accessing PCI Configuration Space, Offset<23:16> defines the PCI Bus Number that is being addressed, Offset<15:11> defines the Device Number, Offset<10:8> defines the Function Number, and Offset<7:2> defines the Register Number.

If the Bus Number for a configuration access matches the Bus Number of the PCI Leaf being accessed, a Type 0 Configuration Cycle will be generated. If the Bus Number is greater than the PCI Leaf's Bus Number, but less than or equal to the PCI Leaf's Subordinate Bus Number, a Type 1 Configuration Cycle will be generated. Otherwise, the transaction will cause an error.

For Type 0 Configuration Cycles, the PCI address that is issued is built as follows:

- PCI<31:11> = 2 ^ DeviceNumber[i.e. PCI<11+DeviceNumber> will be set to 1, all other bits in PCI<31:11> will be set to 0].
- PCI<10:8> = Function Number
- PCI<7:2> = Register Number
- PCI<1:0> = 00

In addition, if DeviceNumber falls within the supported range (1-4 for PCI-A, and 1-6 for PCI-B), one of the IDSEL s from the board will be asserted as follows:

- IDSEL<n:0> = 2 ^ (DeviceNumber-1) [i.e. IDSEL<DeviceNumber-1> will be set to 1, all other bits in IDSEL<n:0> will be set to 0. For PCI-A, n=3, and for PCI-B, n=5].

Example: if Offset<23:0> = 0x121C00, the configuration access is to Bus Number 0x12, Device Number 3, Function Number 4. If the Bus Number of the PCI Leaf is 0x12, a Type 0 Configuration Cycle will be generated with PCI<31:0> = 0x00004400, and IDSEL<n:0> = 0x04.

Note – IDSELS are going to be generated by the board according to the same DEVICE Number -> IDSEL mapping as in SUPCISUPCI. And as long as OBP is notified of the IDSEL -> PCIAD -> Device mapping for the platform, we are o.k.

A PCI Special Cycle is simply a special case of a Type 0 Configuration Cycle, where Offset<15:2> = 0x3fc0.As a convenient way to generate special cycles, JIO uses device 31,function 7 on bus 0. Hence the address offset[15:0] is 0xff00. NOTE: this is the byte offset. This generates special cycles only, no configuration cycles.

For Type 1 Configuration Cycles, the PCI address that is issued is built as follows:

- PCI<31:24> = 00000000
- PCI<23:2> = Offset<23:2>
- PCI<1:0> = 01

See the PCI Specification for more details on the format of Configuration Space addresses.

JIO issues reads and writes to PCI Config pace with proper Byte Enables as requested from Jbus interface. JIO never asserts req64_l for Config read / write accesses. The following are JIO's read and write transaction protocols to PCI Config Space for 32 and 64 bit clients.

- For <= 4 byte config access with 32 / 64 bit client, and byte enables within 32 bit address boundary, JIO does not assert req64_l. It does a single data beat with 32 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.

- For <= 4 byte config access with 32/64 bit client, and byte enables across 32 bit address boundary, JIO does not assert req64_l. It does two data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.
- For > 4 & <= 8 bytes config access with 32/64 bit client, and byte enables within 64 bit address boundary, JIO does not assert req64_l. It does two data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.
- For > 4 & <= 8 bytes config access with 32/64 bit client, and byte enables across 64 bit address boundary, JIO does not assert req64_l. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 1, JIO does 1 PIO transaction with four data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 0, JIO does 2 PIO transactions with 2 data beats in each transaction at 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.
- For > 8 byte config access with 32/64 bit client, JIO does not assert req64_l. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 1, JIO does 1PIO transaction with four data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 0, JIO does 2 PIO transactions with 2 data beats in each transaction at 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.

PCI I/O Space

- PCI<31:24> = 00000000
- PCI<23:0> = Offset<23:0>

JIO issues reads and writes to PCI I/O Space with proper Byte Enables as requested from Jbus interface. JIO never asserts req64_l for I/O read/write accesses, but always issues single data beat with ad[31:0] and cbe[3:0] valid per beat.The ad[31:0] always points to the least significant valid byte enable for each beat. JIO breaks up one Jbus transaction into multiple "single-data-beat" I/O transactions and drops those addresses at 32 bit boundaries that have byte enables from Jbus turned off. The following are JIO's read and write transaction protocols to PCI I/O Space for 32 and 64 bit clients.

- For <= 4 byte I/O access with 32/64 bit client, and byte enables within 32 bit address boundary, JIO does not assert req64_l. It does a "single-data-beat" with ad[31:0] pointing to the least significant valid byte enable.

- For <= 4 byte I/O access with 32/64 bit client, and byte enables across 32 bit address boundary, JIO does not assert req64_l. It does two “single-data-beats “ with ad[31:0] pointing to the least significant valid byte enable for each transaction.
- For > 4 & <= 8 bytes I/O access with 32/64 bit client, and byte enables within 64 bit address boundary, JIO does not assert req64_l. It does two “single-data-beats “ with ad[31:0] pointing to the least significant valid byte enable for each transaction.
- For > 4 & <= 8 bytes I/O access with 32/64 bit client, and byte enables across 64 bit address boundary, JIO does not assert req64_l. It does 2 to 3 single-data-beats “ with ad[31:0] pointing to the least significant valid byte enable for each transaction.
- For > 8 bytes I/O access with 32/64 bit client, and byte enables across 64 bit address boundary, JIO does not assert req64_l. It does 3 to 4 single-data-beats “ with ad[31:0] pointing to the least significant valid byte enable for each transaction.

PCI Memory Space

- PCI<31:2> = Offset<31:2>
- PCI<1:0> = 00

JIO issues reads and writes to PCI memory Space with proper Byte Enables as requested from Jbus interface. The following are JIO’s read and write transaction protocols to PCI Memory Space for 32 and 64 bit clients.

- For <= 4 byte memory access with 32/64 bit client, and byte enables within 32 bit address boundary, JIO does not assert req64_l. It does a single data beat with 32 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don’t care. PCI ad[31:0] always has valid data.
- For <= 4 byte memory access with 32/64 bit client, and byte enables across 32 bit address boundary, JIO does not assert req64_l. It does two data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don’t care. PCI ad[31:0] always has valid data.
- For > 4 & <= 8 bytes memory access with 32/64 bit client, and byte enables within 64 bit address boundary, JIO does not assert req64_l. It does two data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don’t care. PCI ad[31:0] always has valid data.
- For > 4 & <= 8 bytes memory access with 32 bit client, and byte enables across 64 bit address boundary, JIO asserts req64_l. But since the client is 32 bit, it does not receive ack64_l. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 1, JIO does 1 PIO transaction with four data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don’t care. PCI ad[31:0] always has valid data. If ARB_PARK bit [bit 16] in PCI Control and

Status Register is set to 0, JIO does 2 PIO transactions with 2 data beats in each transaction at 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.

- For > 4 & <= 8 bytes memory with 64 bit client, and byte enables across 64 bit address boundary, JIO asserts req64_l. Since the client is 64 bit, it receives ack64_l, and does two data beats with 64 bit address boundary; cbe[7:0] pointing to requested bytes and PCI ad[63:0] has valid data.
- For > 8 byte memory access with 32 bit client, JIO asserts req64_l. But since the client is 32 bit, it does not receive ack64_l. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 1, JIO does 1 PIO transaction with four data beats with 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data. If ARB_PARK bit [bit 16] in PCI Control and Status Register is set to 0, JIO does 2 PIO transactions with 2 data beats in each transaction at 64 bit address boundary; cbe [3:0] points to the requested bytes, cbe[7:4] are don't care. PCI ad[31:0] always has valid data.
- For > 8 byte memory access with 64 bit client, JIO asserts req64_l. Since the client is 64 bit, it receives ack64_l, and does two data beats with 64 bit address boundary; cbe[7:0] pointing to requested bytes and PCI ad[63:0] has valid data.

The following table summarizes JIO's PCI transaction protocols for PIO accesses to Memory, I/O and Config Spaces in a tabular form

TABLE 3-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Config	<= 4	32/64 bit	within 32 bit address boundary	No	1 , with 32 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	<= 4	32/64 bit	across 32 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 3-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Config	> 4 and <= 8	32/64 bit	across 64 bit address boundary	No	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	> 8	32/64 bit	across 64 bit address boundary	No	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	<= 4	32/64 bit	within 32 bit address boundary	No	1 single-data-beat, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 3-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
I/O	<= 4	32/64 bit	across 32 bit address boundary	No	2 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	> 4 and <= 8	32/64 bit	across 64 bit address boundary	No	2 to 3 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	> 8	32/64 bit	across 64 bit address boundary	No	3 to 4 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	<= 4	32/64 bit	within 32 bit address boundary	No	1 , with 32 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 3-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Mem	<= 4	32/64 bit	across 32 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 4 and <= 8	32 bit	across 64 bit address boundary	Yes (but no ack64)	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 3-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Mem	> 4 and <= 8	64 bit	across 64 bit address boundary	Yes (gets ack64)	2, with 64 bit address boundary	[7:0]point to requested bytes	ad[63:0] has valid data
Mem	> 8	32 bit	across 64 bit address boundary	Yes (but no ack64)	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 8	64 bit	across 64 bit address boundary	Yes (gets ack64)	2, with 64 bit address boundary	[7:0]point to requested bytes	ad[63:0] has valid data

3.2.2.3 Jbus Interrupt Space

JIO will ignore any interrupt sent to it by mistake.The system will hang eventually if that happens.

3.3 UPA => Jbus

The UPA64S is a slave only interface, so there are no transactions for JIO to respond to on this bus.

3.4 PCI => Jbus

PCI-A and PCI-B are identical in terms of how the PCI address spaces are handled; this section applies to both.

3.4.1 PCI Configuration Space

JIO responds to Configuration Read or Configuration Write cycles on the PCI bus to only two registers in the PCI Leaf block :

i) Extension Register (section 4.5.1.9: address 0x50)

ii) PCI Target Address Space Register (section 4.5.1.7: address 0x54)

Accesses from the Jbus that target all other configuration registers that are internal to a PCI Leaf Block will be serviced by the PCI Leaf without generating a configuration cycle on the PCI bus.

3.4.2 PCI I/O Space

JIO does not respond to any PCI I/O space transactions (I/O Read or I/O Write command types).

Action Item: Check Sabre manual for Error handling

3.4.3 PCI Memory Space

This is the space in which DVMA, and DMA (IOMMU bypass) activity take place. PCI peer-to-peer activity may also use this address space, and is included in the tables below, although JIO is not involved in these transfers.

3.4.3.1 PCI Address Translation Modes

The final destination and address translation of a PCI Memory transaction is based on:

- PCI addressing mode used: 64-bit (DAC) vs. 32-bit (SAC)
- PCI address bit <31> in SAC mode
- Value of MMU_EN in the IOMMU Control Register
- Value of PCI address bits <63:50> in DAC mode
- Hit on PCI Target Address Space Register

In all cases, JIO will only support bursts as a target device in Linear Incrementing mode (i.e. PCI<1:0> must be 00). If any of the reserved modes are used, JIO will only transfer a data phase, and then issue a target disconnect.

The following table shows the various ways that JIO as a PCI target device deals with PCI Memory Space addresses.

TABLE 3-4 PCI DVMA Modes of Operation

Mode	Hit in PCI Target Address Space register	MMU_EN	Addr<63:50>	Extension Enable Bit (extension reg[11])	Result
SAC	0	X	N / A	X	PCI peer-to-peer (Ignored by JIO)
SAC	1	0	N / A	0	Pass-through. (Not implemented)
SAC	1	0	N / A	1	Extension Mode
SAC	1	1	N / A	X	IOMMU Translation (DVMA)
DAC	X	X	0x0000-0x3FFE	X	Ignored by JIO
DAC	X	X	0x3FFF	X	Bypass (DMA)

Pass-through

In pass-through mode, Jbus<42:31> = 0x000, Jbus<30:4> = PCI<30:4>. Pass-through transfers always generate cacheable transactions on Jbus.

Note – JIO has not implemented the Pass-through Mode. In case Jbus addresses need to be generated for SAC from PCI with upper bits = 0, the extension mode can be used with the extension register programmed with zeroes . The extension mode gives maximum flexibility to generate any Jbus address as desired . This implies that to get proper Jbus addresses coming out from JIO for PCI SAC accesses, MMU and Extension Mode *cannot* both be disabled.

Extension Mode

In order to enable one JIO to configure another JIO without the CPUs present (test mode bringup), JIO supports the Extension Mode of address translations in which while the IOMMU is disabled, SAC, and Extension Enable = 1, Jbus<42:32> = Extension Register<10:0>, and Jbus<31:0> = PCI<31:0>.Please refer to section 4.5.1.8 for the details on the Extension Register.

This mode would enable JIO to generate all possible Jbus addresses, without initializing the IOMMU, using only 32 bit single address cycle PCI transactions.This allows, for instance, another JIO or another CPU to be exercised.

IOMMU Translation Mode

In IOMMU translation mode, the physical address is obtained by performing a virtual to physical translation through the IOMMU. The value of the cacheable (C) bit in the TTE for the virtual page determines whether the Jbus transaction generated is cacheable or noncacheable.

PCI Peer-to-Peer Mode

In peer-to-peer mode, two devices on the same PCI bus transfer data without any involvement from JIO. The master device simply puts out the PCI address to which the target device has been mapped. Whether there is any subsequent address translation involved is device dependent. If no device has been mapped at the target address, the PCI master device will terminate its cycle with a Master-Abort.

Bypass Mode

In bypass mode, the Jbus<42:4> = PCI<42:4>. A cacheable Jbus transaction will be generated if PCI<42> = 0, otherwise a noncacheable transaction type will be used.

3.4.3.2 Jbus Transactions Generated

The exact Jbus transaction(s) initially generated for a given PCI DMA transaction depends upon:

- Transfer direction: read / write
- Cacheability (C), as documented above.
- Transfer size.

The transfer size can be a complete cache line (64 bytes), or a partial cache line, in which case the size refers not to the number of valid bytes, but to the number of bytes spanned by the valid bytes, and then extended to the nearest 8-byte aligned boundary on either side (e.g a transfer in which only bytes 31 and 32 are valid has a 16-byte transfer size)

For noncacheable transactions the transfer size is determined solely by the burst length on the PCI bus and the byte enables of each data phase.

For cacheable transactions, the transfer size is always 64 bytes.

Note that this table does not show all of the potential Jbus transactions that may be issued in servicing a PCI DMA request..

TABLE 3-5 Jbus Transactions generated for PCI DMA activity

C	R/W	Size (bytes)	Jbus Result(s)
1	Read	Any	RDS, RDSA, RD, RDD, RDO, OWN
1	Write	64	WRI, WRIS
1	Write	Partial 0-64	RD* (RDS,RDSA,RD,RDD,RDO,OWN) & WRBK (only if dirty)
0	Read	Any	NCRD, NCBRD
0	Write	64	NCBWR
0	Write	Partial 0 -16	NCWR

3.4.4 Interrupts

For any interrupts generated by the PCI leaf (this includes all interrupts that JIO is currently capable of generating), a Jbus INT transaction will be issued by JIO, using the following Jbus address:

- J_AD<35:0> = Reserved, issued as 0.
- J_AD<40:36> = JPID of Target Port
- J_AD<42:41> = Reserved, issued as 0
- J_AD<47:43> = 0x14, Jbus Transaction Type = INT
- J_AD<63:48> = Reserved, issued as 0
- J_AD<127:64> = Same as J_AD<63:0>

3.5 Little-endian Address Spaces

3.5.1 Overview

The main interface of JIO, the Jbus, is big-endian. Other interfaces, notably the two PCI buses, are little-endian. JIO provides the necessary support to connect the two together in a consistent fashion. The main feature is called “byte twisting”: from a hardware perspective, the bytes on the datapath from a PCI bus are twisted around before connecting to any other datapath within JIO, so that bits 63:56 map to bits 7:0, bits 55:48 map to bits 15:8, etc. From another perspective, this ensures that logical byte lanes are connected: the byte at address 0 on the big-endian side is directly wired to the byte at address 0 on the little-endian side. As a result, all byte-sized PIOs and byte-stream DMA is handled correctly. This, along with other features built into SPARC V9 processors, provides a mechanism for all PIO and DMA activity to/from the PCI bus to take place correctly.

3.5.2 Big- and Little-endian Regions

3.5.2.1 Address Space

JIO responds to several different regions of noncacheable address space on the Jbus. As indicated there, the following regions are little-endian, and all accesses to them use byte twisting.

- ° PCI-A_MemBase
- ° PCI-A_ConfigBase
- ° PCI-A_IOBase
- ° PCI-B_MemBase
- ° PCI-B_ConfigBase
- ° PCI-B_IOBase

All other address regions are big-endian, and there is no byte twisting done for accesses other than for the regions listed above.

3.5.2.2 Internal Blocks

Given the above breakdown of address regions, most of JIO's internal blocks are in big-endian address space: the 64-bit data paths within the JIO design blocks are connected to the JIO's Jbus data bus with no byte twisting. The exceptions are the PBM blocks within each PCI leaf. Since the PBM controls the little-endian PCI bus, it is considered to be a little-endian block. For each data interface entering/leaving a PBM block, byte twisting is in effect (in addition to interfacing to PIO and DMA buses within the PCI leaf, each PBM has data interfaces with its associated streaming cache block).

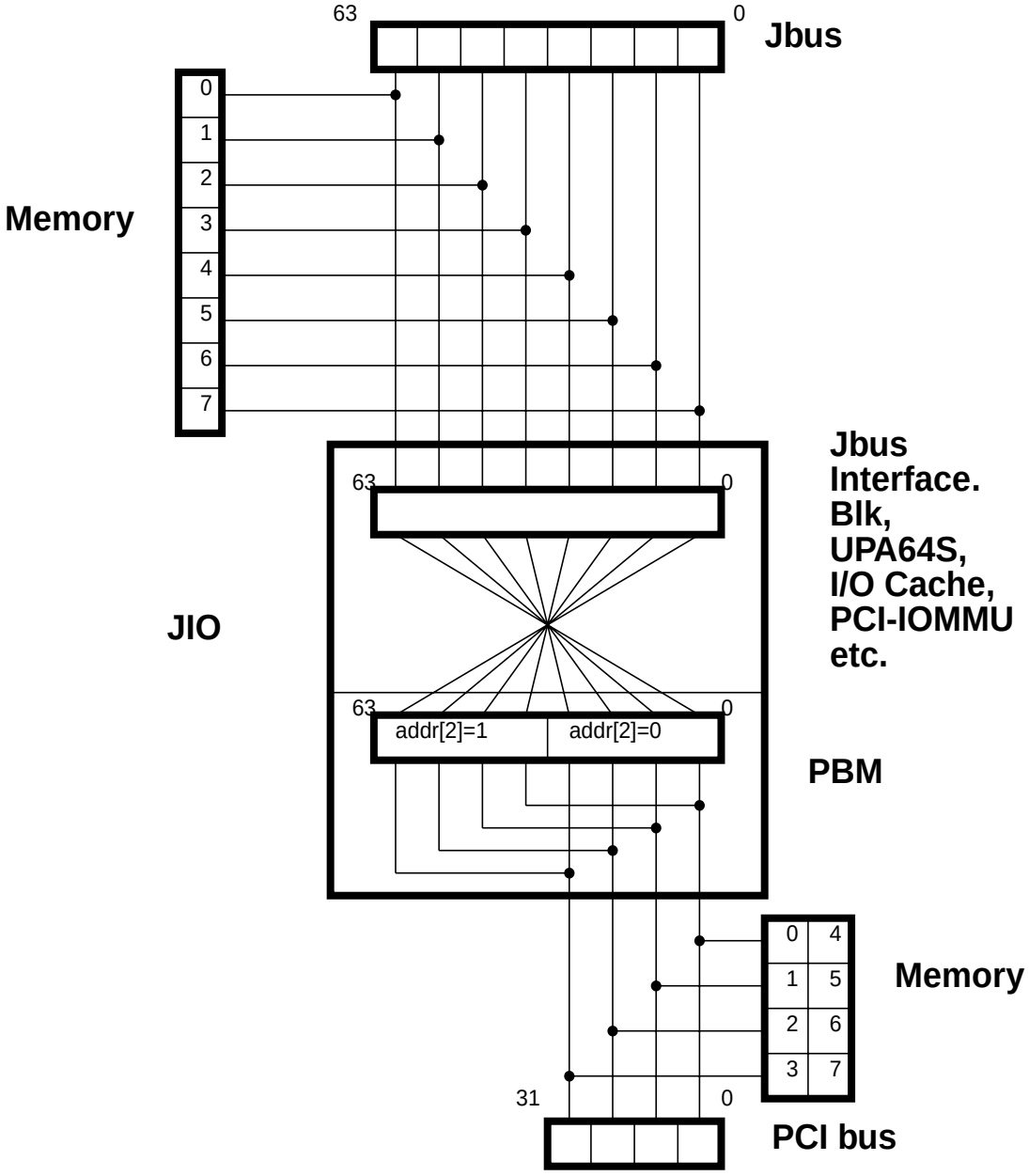
Note – Each PBM also contains some internal control/status registers mapped into big-endian spaces (PCI-A_CSRBase and PCI-B_CSRBase). So that these registers are not affected by the byte twisting of the PBMs data paths to the rest of the chip, within the PBM, the data path to these registers is "retwisted".

3.5.3 Byte Twisting

Figure 3-2 diagrams what is meant by byte twisting. It shows how data is manipulated from a 32-bit little-endian PCI bus to a 64-bit big-endian Jbus. The case of a 64-bit PCI bus is a straightforward modification of this diagram, and won't be shown.

For each bus, a typical connection to memory is shown, along with the byte addresses of the memory. This is mainly for reference - it is one way of showing exactly what is meant by big- or little-endian. It helps to show that the "logical" byte lanes of each bus are correctly connected through JIO.

FIGURE 3-2 JIO Byte Twisting



3.5.4 Specific Cases

The following sections detail specific types of data transfers, and how correct byte ordering is maintained in each case.

3.5.4.1 Programmed Input/Output (PIO)

Normal

Due to the byte twisting, all byte sized PIOs work correctly. The byte lane used for a given address on the big-endian side is directly wired to the byte lane used for that address on the little-endian side.

For any access larger than a byte, byte-twisting is not sufficient. For example, if the 32-bit value 0x12345678 is written to a 32-bit register on a PCI device, the PCI device will interpret the value as 0x78563412 after byte twisting.

To correct for this, SPARC V9 CPUs have special support for little-endian access. By either marking the page containing the PCI register as little-endian in the processor's MMU, or by using one of the little-endian Address Space Identifiers (ASIs), the CPU will alter its ordering of the bytes, in effect undoing the byte-twisting that will be done by JIO, so that the PCI device correctly sees 0x12345678.

Note – In case any one of the little-endian registers in JIO is 32 bits wide, then we need to choose the proper 32 bit half of the Jbus 128 bit data for the CSR write by looking at PA [3,2].

3.5.4.2 Direct Memory Access (DMA)

Data Streams

Because byte lanes at the same address are connected, DMA of byte streams works correctly without any further intervention. A PCI device that receives the byte stream 0x01 0x02 0x03 0x04 would pack the bytes into a 32-bit register starting with the LSB of the register, i.e. 0x04030201. If this were transferred to memory on the PCI bus, the value 0x01 would be at the lowest memory location, as desired.

After byte twisting, the value that appears on the Jbus would be 0x01020304. Since the MSB on the Jbus is the lowest memory location, the value 0x01 is still stored at the lowest memory location, as desired.

Descriptors

This case is similar to PIOs of size greater than one byte. With just byte twisting, a DMA descriptor access would get the wrong byte ordering. For example, if the value 0x12345678 were set up in an address field in a descriptor, a PCI device using DMA to fetch the descriptor would see the value as 0x78563412 instead.

To avoid this, the little-endian features of the processor are used again. Processor loads and stores to the descriptors should be specified as little-endian. This will re-order the bytes in memory when the descriptor is built so that after byte twisting, the PCI device sees the correct value.

CHAPTER 4

Configuration and Status Registers

The JIO Configuration and Status registers (CSRs) are described in this chapter. The following abbreviations are used throughout this chapter:

- **R** - *Read only: a register field that is not writable*
- **R/W** - *Read/write: A standard register field*
- **R/W1C** - *Read / write 1 to clear: Writing a 0 to bits in this field has no affect, but writing a 1 to a bit in this field will cause that bit to be set to 0.*
- **W** - *Write only: reads of this field return undefined data*
- **DMA** - *Direct Memory Access: A transaction initiated by a leaf block resulting in a Jbus transaction*
- **DVMA** - *Direct Virtual Memory Access: A subclass of DMA transactions in which the address of the leaf transaction is treated as a virtual address and translated by an MMU.*

4.1 General Information

4.1.1 Access Size

Each register in JIO has a natural access size, which is documented below. For most registers this size is 8 bytes, but there are exceptions. Register accesses should always be done with the indicated access size, or undefined behavior may result, including, but not limited to:

- Incorrectly sized writes may still write data to the entire register
- Incorrectly sized reads may not trigger a side-effect (if any) of the register
- Reads or writes that are too large (span multiple registers) can return incorrect data or corrupt any of the addressed registers.

4.1.2 Unimplemented Addresses

Any address within JIO's register address space that is not documented here as belonging to a specific register is reserved and should not be read or written by software. Although, JIO will "own" and respond to its entire 128 Gbyte address range, which is the allocated space for each agent attached to Jbus (64 Gbyte of cacheable space, and 64 Gbyte of non-cacheable space, a read to a unimplemented address will return undefined data and a write will simply be ignored by JIO. Since JIO will cover its entire allocated address space, system hangs due to accesses to unimplemented addresses will be reduced, although this behavior should still be avoided by software.

4.1.3 Physical Addresses

Complete Jbus physical addresses are not shown in here for any registers, since each address region, except Jbus_Base, that JIO responds to is directly relocatable. The physical address for Jbus_Base is:

```
{1,0,13'b0, JPID[4:0], 23'b0}
```

where JPID[4:0] is JIO's Jbus Port ID.

Instead, all register addresses are documented as offsets within a particular address region. Unless otherwise indicated, all offsets are with respect to the Jbus_Base region (this accounts for almost all of JIO's internal registers). For more details about JIO's address regions, see Chapter 3.

4.2 Jbus Interface

TABLE 4-1 Jbus Register Offsets

Register	Offset	Access Size
Jbus Device ID Register	0x00.0000	8 bytes
UPA0 Offset Base Register	0x40.0000	8 bytes
UPA0 Offset Mask Register	0x40.0008	8 bytes
UPA1 Offset Base Register	0x40.0010	8 bytes
UPA1 Offset Mask Register	0x40.0018	8 bytes

TABLE 4-1 Jbus Register Offsets

Register	Offset	Access Size
NewLink Address Match Register	0x40.0020	8 bytes
NewLink Address Mask Register	0x40.0028	8 bytes
NewLinkAlt Address Match Register	0x40.0030	8 bytes
NewLinkAlt Address Mask Register	0x40.0038	8 bytes
PCI-A_Mem Offset Base Register	0x40.0040	8 bytes
PCI-A_Mem Offset Mask Register	0x40.0048	8 bytes
PCI-A_Cfg_IO Offset Base Register	0x40.0050	8 bytes
PCI-A_Cfg_IO Offset Mask Register	0x40.0058	8 bytes
PCI-B_Mem Offset Base Register	0x40.0060	8 bytes
PCI-B_Mem Offset Mask Register	0x40.0068	8 bytes
PCI-B_Cfg_IO Offset Base Register	0x40.0070	8 bytes
PCI-B_Cfg_IO Offset Mask Register	0x40.0078	8 bytes
JIO Control/Status Register	0x41.0000	8 bytes
Jbus Error Control Register	0x41.0008	8 bytes
Jbus Interrupt Control Register	0x41.0010	8 bytes
Jbus Error Log Register	0x41.0018	8 bytes
Jbus Parity Control Register	0x41.0020	8 bytes
UE AFSR	0x41.0030	8 bytes
UE AFAR	0x41.0038	8 bytes
CE AFSR	0x41.0040	8 bytes
CE AFAR	0x41.0048	8 bytes
Jbus Energy Star Control Register	0x41.0050	8 bytes
Jbus Change Initiation Register	0x41.0058	8 bytes
Queue Control Register	0x41.1000	8 bytes
Jbus DTag Diagnostic Registers	0x41.2000 - 0x41.2070	8 bytes
Jbus CTag Diagnostic Registers	0x41.3000 - 0x41.3070	8 bytes
Safari Debug Registers	0x41.4000 - 0x41.4018	8 bytes
Jbus Performance Control Register	0x41.7000	8 bytes
Jbus Performance Counter Register	0x41.7008	8 bytes
Reset_Gen Register (NEW)	0x41.7010	8 bytes

TABLE 4-1 Jbus Register Offsets

Register	Offset	Access Size
Reset_Source Register (NEW)	0x41.7018	8 bytes
UPA Reset Control Register(NEW)	0x41.7020	8 bytes
GPIO 0 Register	0x46.0000	1 byte
GPIO 1 Register	0x46.0001	1 byte
GPIO 2 Register	0x46.2000	1 byte
GPIO 3 Register	0x46.2001	1 byte
GPIO Data Register	0x46.4000	8 bytes
GPIO Control Register	0x46.4008	8 bytes

Note – The shaded registers are reserved for JIO. Any reads to these addresses would return undefined data, any write is ignored.

Note – The reset-related registers are described in detailed in Chapter 7 "Reset".

Note – All of the registers mentioned above other than the GPIO registers need to be written to with addresses aligned at 64 bit boundary. JIO ignores writes to these registers if the write is not aligned at 64 bit address boundary.

4.2.1 Jbus Device ID Register

TABLE 4-2 Jbus Device ID Register(Jbus_Base + 0x00.0000)

Bits	Field	Description	Default Value	Type
63:56	Cookie	This field is provided so that the open boot PROM code detects that JIO does not support an Fcode PROM.	0xFC	R
55:34	Rsvd	Reserved. Read as zero	0x00	R
33:27	Jbus Ports (NEW)	Valid Jbus ports in current system configuration. Each bit of this field is used to detect the presence of a Jbus Port in the system. If set to one, the Jbus Port is invalid and if zero, the Jbus Port is valid. The value of J_PACK[2:0] for each port is sampled after reset deassertion, if DTL pull-up on the J_PACK pins indicate “pulled up” state, the corresponding port is considered to be invalid for the current configuration.	-	R
26:22	Rsvd	Reserved. Read as zero	0x00	R
21	JPID[4]	Most significant bit of JIO’s Jbus Port ID.	1	R
20:17	JPID[3:0]	Least Significant 4bits of JIO’s Jbus Port ID. The Port ID is read-only here, and bit “1” is set during reset from external j_id[1] pin on JIO.	-	R
16	M/S	Master/Slave. JIO is both a master and slave device on Jbus	1	R
15:10	MID	Manufacturer ID.	0x23	R
9:4	MT	Module Type.	0x10	R
3:0	MR	Module Revision : 0x0 for JIO 1.0 0x1 for JIO 2.0 0x2 for JIO 2.1	See Description	R

4.2.2 Offset Base and Mask Registers

JIO is using a common implementation for all of its address match and mask registers although in some cases this leads to extra bits that are not strictly necessary as they allow JIO to map a region larger than what is implemented by the corresponding I/O bus (UPA64S or PCI).

JIO at first checks PA bits [42:36] ([42:41] == 11, [40:36] == JPID) to see if it should respond to the Jbus Address. If the address is found to be within its memory range, bits [35:24] are masked by Offset_Mask register bits [35:24] and compared against the Offset_Base registers bits[35:24].

The address filtering is performed as described by the picture below:

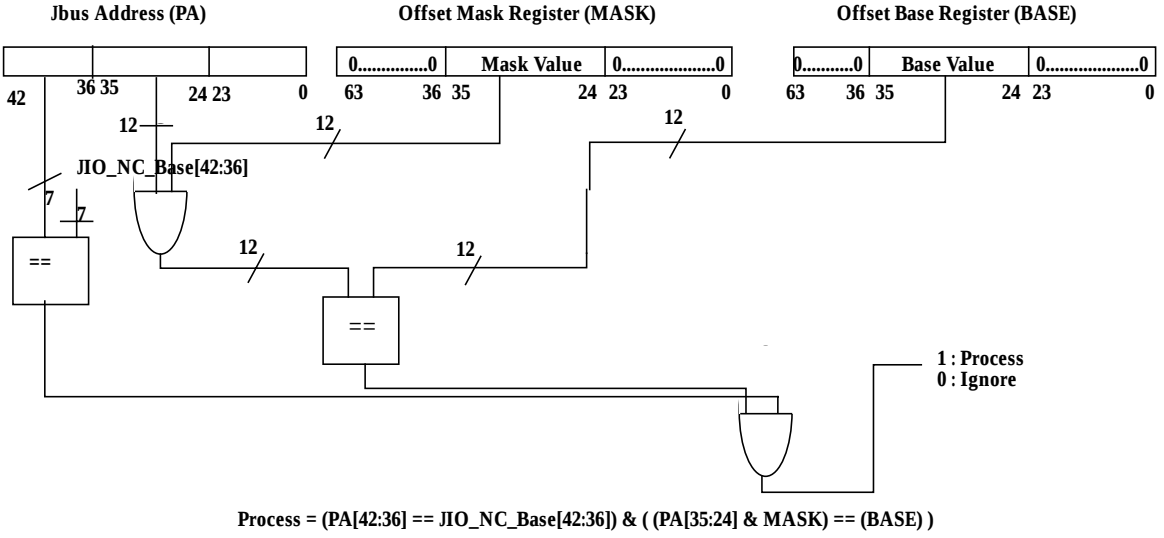


FIGURE 4-1 Address Filtering in JIO

When a Jbus request is filtered positively it is forwarded to the appropriate block (UPA, PCI A or PCI B) with the most significant bits of the Jbus address (for which Mask bits = 1's) chopped off.

The format of the Offset Base registers is:

TABLE 4-3 Offset Base Register(Jbus_Base + 0x40.0000 / 0x40.0010 / 0x40.0040 / 0x40.0050 / 0x40.0060 / 0x40.0070)

Bits	Field	Description	Reset Value	Type
63	V	Valid. Hardwired to 1.	-	R
62:36	Rsvd	Reserved. Read as zero.	0x0 0000	R
35:24	Base	Base Value. This is the value of the base address to be matched against.Depending on the Mask Bits,software should load Base bits corresponding to Mask bits = 0 with all zeroes.	*See note	R / W
23:0	Rsvd	Reserved. Read as zero.	0x00 0000	R

Note – * Reset State of Offset Base Register bits [35:24] is 0x000 for UPA0,UPA1, PCIB Mem, PCIA Config / IO and PCIB Config / IO address spaces. For PCIA Mem space however, the reset value of Offset Base Register bits [35:24] is **0xF00** to enable access to the Boot PROM at Power-On through the Master JIO.

The format of the Offset Mask registers is:

TABLE 4-4 Offset Mask Register(Jbus_Base + 0x40.0008 / 0x40.0018 / 0x40.0048 / 0x40.0058 / 0x40.0068 / 0x40.0078)

Bits	Field	Description	Reset Value	Type
63:43	RESERVED	Reserved, read as 0.	0x00.0000	R
42:36	MASK_HI	High order mask bits, always set to 1s.	0x7F	R
35:24	Mask	Mask Value. This mask is used to remove the least significant bits before matching. It defines the size of the mapped region. It must be string of 1 followed by a string of 0.	*See note	R / W
23:0	Rsvd	Reserved. Read as zero	0x00 0000	R

Note – * Reset State of Offset Mask Register bits [35:24] is 0xF80 for UPA0,UPA1, PCIB Mem, PCIA Config / IO and PCIB Config / IO address spaces. For PCIA Mem space however, the reset value of Offset Mask Register bits [35:24] is **0xF00** to enable access to the Boot PROM at Power-On through the Master JIO.

Each pair of Offset Base and Mask register can map a space between 16 MB and 64 GB, although not all possible sizes make sense. TABLE 4-5 lists the maximum, minimum and recommended sizes for each address space. Internally, JIO will ignore

TABLE 4-5 Offset Base / Mask size limitations

Address Space	Min Size	Max Size	Recommended		Comments
			Size	Mask[35:24]	
UPA0 UPA1	2GB	8GB	8GB	0xE00	Existing UPA framebuffers require 2GB, future devices may require up to 8GB. UPA does not support >8GB per port.
PCI-A_Mem PCI-B_Mem	16MB	4GB	2GB	0xF80	Limits amount of slave space available for mapping PCI devices. PCI doesn't support more than 4GB, while JIO normally limits slave space to 2GB at most.
PCI-A_Cfg_IO PCI-B_Cfg_IO	32MB	32MB	32MB	0xFFE	If set to 16MB, either PCI Config transactions or PCI I/O transactions will be disabled (determined by value of Match[24]).

any address bits beyond the maximum size listed here. If programmed to a larger size, the given address space will be aliased at the Max Size boundaries.

Note – The timing of changes to these registers when a PIO write is performed is somewhat indeterminate. If software wants to ensure that a change takes effect before proceeding, it should follow the PIO write by a PIO read of these registers, before sending writes and reads to the configured address space.

4.2.3 JIO Control/Status Register

TABLE 4-6 JIO Control/Status Register(Jbus_Base + 0x41.0000)

Bits	Field	Description	Reset Value	Type
63:54	Rsvd	Reserved. Read as zero.	0x00	R
53	Disable_M_ State	JIO M state cache install enable/disable mode bit. Valid encodings are : 0 : M state install enabled ; JIO can install lines in M state in its I/O caches. JIO then performs partial-line writes directly to these lines from PCI . 1 : M state install disabled; JIO cannot install lines in M state in its I/O caches.Instead it installs in O state for all the cases where it would have installed in M state. JIO subsequently issues OWN transaction on JBus in order to perform partial-line stores to those lines from PCI. Only one of these two encodings/modes are valid in any given system but not both. So depending on the mode desired, OBP should program this bit accordingly and never change it . The two main differences in functional behavior of JIO between the two modes visible at a system level are : i) JIO would preserve Store-Store order between Full line writes and Partial line writes in memory when Disable_M_State = 1. No Software intervention would be needed to preserve order as in the case of Disable_M_State = 0. ii) an increased number of “OWN” JBus transactions would be issued by JIO when Disable_M_State = 1.	0	R/ W
52:47	JBUS_PLL_ TUNE[5:0]	Tune bits for Jbus PLL.They are used to modify the PLL loop parameters to compensate for high levels of noise or input jitter.This TUNE bit programming capability provide the flexibility for the system engineer to optimize the PLL stability and jitter. Normally, the PLL must be reset when this field is changed by software. The only exception to this (and the recommended way of changing this field without causing a PLL reset) is to change the tune bits 1 LSB at a time, with a hold of 1 microsecond at each step. For instance, to move from TUNE(5:0)=’010010’ to TUNE(5:0)=’010101’ would require the following sequence, and a hold of 1us at each point. Initial TUNE(5:0)=’010010’ Mid-1 TUNE(5:0)=’010011’ Mid-2 TUNE(5:0)=’010100’ Final TUNE(5:0)=’010101’	0x12	R/ W

TABLE 4-6 JIO Control/Status Register(Jbus_Base + 0x41.0000)

Bits	Field	Description	Reset Value	Type
46:45	L5CLK_Sel	L5CLK select. Valid encodings are: 00 : Jbus internal clock driven out on L5CLK 01 : 1'b0 driven on L5CLK 10 : Primary PCI internal clock driven out on L5CLK 11 : UPA/Secondary PCI internal clock driven out on L5CLK	0x0	R/W
44	Par_Delay	Delay internally generated control parity by one clock if set to 1. This is needed to align the internally generated control parity with the J_PAR on the pin which can come through repeaters in the system with multiple Jbus segments, before comparing against J_PAR.	0	R/W
43	Control Parity Error Detection Enable (NEW)	Control Parity Error Check Enable. The bit encodings are: 0 : Control Parity Error Check Disabled. 1 : Control Parity Error Check Enabled.	0	R/W
42:36	Jpack_Delay [6:0]	J_pack delay control bit for each port. Bit 0 corresponds to J_PACK0, bit 6 to J_PACK6. the control bit encodings are: 0 : Do not delay J_PACK for that port into the parity check/generate logic. Bit should be 0 if Jbus port is on a remote Jbus segment. 1 : Delay J_PACK by one clock for that port into the parity check/generate logic. Bit should be 1 if Jbus port is on a local Jbus segment. Note: Reset state is 0x7f. So the implication is at reset, all Jbus ports should have the control parity check disabled. JIO has its own parity check disabled at reset. Have to make sure that the CPUs also do the same thing. Once software comes and updates this register in all Jbus ports, then all the ports should have their control parity error detection enable turned on, other than JIO with JPID[2:0] == 6, which should always drive J_PAR and not detect control parity error.	0x7f	R/W

TABLE 4-6 JIO Control/Status Register(Jbus_Base + 0x41.0000)

Bits	Field	Description	Reset Value	Type
35:34	DTL Mode	DTL mode for Driver/Receiver group. The Mode Encodings are: 0x0 - Reserved 0x1 - DTL-end mode 0x2 - DTL-mid mode 0x3 - DTL-2 mode This field is read-only here. The DTL modes are set upon reset from 2 external hardwired pins in JIO as per the following relationship: dtl_mode = {j_down25, ~j_upopen}, where: j_down25 = 25 ohm pull down enable j_upopen = 50 ohm pull-up disable	-	R
33:32	JTO	Jbus watchdog timer timeout interval. Values are: 0x0 => ∞ (Jbus timeouts disabled) 0x1 => 2^{28} Jbus clock cycles (normal) 0x2 => 2^{15} Jbus clock cycles (debug) 0x3 => 2^{9} Jbus clock cycles (simulation) This controls the duration of each Jbus timeout event defined in TABLE 4-10. See cautions on changing JTO while any timeout events are enabled below.	0x0	R/W
31:30	Reserved	Reserved. read as zeroes.	0x0	R
29	Control Parity Gen/Detect Mode (NEW)	Control Parity Generate/Detect Mode.The Read-Only valid encodings are: 0: Generate Control Parity 1: Detect Control Parity In a 2-JIO system, the JIO with JPID[2:0] == 6 (master JIO) will always generate J_PAR after reset deassertion, and the JIO whose JPID[2:0] == 4 (slave JIO) will always detect Control Parity Error if control parity error detection is enabled (bit 43 of JIO Control/Status register is set).	0 for Master Tomatillo. 1 for Slave Tomatillo.	R

TABLE 4-6 JIO Control/Status Register(Jbus_Base + 0x41.0000)

Bits	Field	Description	Reset Value	Type
28:27	Arbitration Mode (NEW)	Arbitration Mode for DEAD cycle in between switching drivers. The valid encodings are: 00: One dead cycle between driver switch. All jbus ports sample J_ADTYPE,J_AD, and J_ADP during dead cycle. This is the only mode for the system with multiple Jbus segments. During dead cycle, termination logic pulls up J_AD, J_ADTYPE and J_ADP to all 1's. 01: One dead cycle between driver switch. No Jbus port samples J_ADTYPE, J_AD, and J_ADP during dead cycle. This mode is used only for the highest frequency single-bus systems, where the bus is operating at such a high frequency that dead cycles do not meet timing.During dead cycle, Jbus is HighZ. 10: No dead cycle between driver switches. All jbus ports sample J_ADTYPE,J_AD, and J_ADP during all cycles. 11 : Reserved.	00	R/W
26:25	Reserved.	Reserved. Read as Zeroes.	0x00	R
24	JPID[4]	Most significant bit of JIO's Jbus Port ID. Programmable by software.	1	R/W
23:20	JPID[3:0]	Least Significant 4bits JIO Port ID. The Port ID is read-only here, and bit "1" is set during reset from external J_ID[1] pin on JIO.	{1,1, j_id[1], 0}	R
19:0	Rsvd	Reserved. Read as zero.	0x00	R

4.2.3.1 Jbus Timeout

Jbus interface will have two watchdog timers for the PCI and GR ports which can be enabled into any one of the "enabled" states through software, or may be disabled. Each timer would start counting when a new read address is written into the Outstanding Read Queue for the corresponding port in Jbus Interface and would reset when the datum for that read is returned to JIO over Jbus. It would start counting immediately after going to reset, provided the Outstanding Read Queue has a new pending read, and reset when the data item for the pending read comes. It would indicate overflow or timeout when the counter reaches the maximum count indicated by JIO Control/Status Register bits [33:32], with the data have not been returned yet.

In case the Counter times out, hardware should log the error, reset the counter and start counting again provided there is at least one more read pending. This would prevent a system hang in the case that multiple transactions to the same address range might all need to have a timeout to make progress.

Timeouts may be changed between “disabled” (0x0) and any “enabled” state without causing spurious timeouts. When in the “disabled” state timeouts are frozen, the interval is literally infinite.

Caution – Changing the timeout interval between “enabled” states may cause spurious timeouts. Changing the timeout interval from one “enabled” state, to the “disabled” state, and then to a different “enabled” state may timeout in the shorter of the two intervals.

4.2.3.2 JIO’s Jbus Port ID

Caution – Changing the JPID[4] field in the JIO Control/Status register causes the location of JIO’s internal registers, PCI Config/IO/Mem Spaces and UPA Mem Spaces to move in physical address space. The timing of this change is imprecise. A subsequent access using the old address may still work, while a subsequent access with the new address may fail.

The following sequence may be used to update the JPID[4] field (assuming that loads are blocking):

- store @old_CSR_address, new_JPID[4]
- load @old_CSR_address (this load may receive an unmapped error)
- load @new_CSR_address (this load should work unless a real error occurs)

Because of the delays associated with writing to a JIO register, the potential error in the above sequence will typically not occur, although it is possible under some conditions.

4.2.4 Jbus Error Control/Log Registers

The Jbus Error Control Register controls which of several possible Jbus errors will result in the assertion of the J_ERR signal by JIO on the Jbus. The J_ERR signal is useful for lab debug in an analyzer. The Jbus Interrupt Control Register controls which of the same set of possible Jbus errors will cause JIO to issue a Jbus Error Interrupt, and the Jbus Error Log Register logs which of the Jbus errors have been detected.

All three registers share a similar format. The common portion is detailed in TABLE 4-10.

TABLE 4-7 Jbus Error Control Register(Jbus_Base + 0x41.0008)

Field	Bits	Reset	Description	Type
ERREN	63	0	Global error report enable for all Jbus specific errors shown in Table 4-10. When set to 0, no Jbus specific errors are reported on J_ERR . When set to 1, Jbus specific errors enabled in the rest of this register are reported on J_ERR.	R/W
Individual Error Enables	62:0	-	See TABLE 4-10 for bit assignments. Reserved bits are read only, all others are R/W. When any of these bits are set to 1, JIO will assert J_ERR if the associated Jbus specific error is detected and ERREN is also 1. The value of these enables is persistent across reset. Error enables for Error bits [21:14] in Table 4-10 are not implemented and writes to those bits would be simply ignored. Reads would return zeroes	R/W

TABLE 4-8 Jbus Interrupt Control Register(Jbus_Base + 0x41.0010)

Field	Bits	Reset	Description	Type
JE_INTEN	63	0	Global error interrupt enable for all Jbus specific errors shown in Table 4-10. When set to 0, no Jbus specific errors are reported via a Jbus Error Interrupt. When set to 1, Jbus specific errors enabled in the rest of this register are reported as a Jbus Error Interrupt.	R/W
Individual Interrupt Enables	62:0	-	See TABLE 4-10 for bit assignments. Reserved bits are read only, all others are R/W. When any of these bits are set to 1, JIO will issue a Jbus Error Interrupt if the associated error is detected and JE_INTEN is also 1. The value of these enables is persistent across reset. Interrupt enables for Error bits [21:14] in Table 4-10 are not implemented and writes to those bits would be simply ignored. Reads would return zeroes.	R/W

TABLE 4-9 Jbus Error Log Register(Jbus_Base + 0x41.0018)

Field	Bits	Reset	Description	Type
ERR_OUT	63	-	Error Out Asserted. This bit is set to a 1 anytime JIO asserts J_ERR. It is persistent across reset, and can only be cleared by writing to this register.	R/ W1C
Individual Error Logs	62:0	-	See TABLE 4-10 for bit assignments. Reserved bits are read only, all others are R/ W1C. JIO will unconditionally set a log bit to 1 when the associated Jbus specific error is detected, regardless of the state of error and interrupt enables. The value of these enables is persistent across reset. Note that for all the snoop errors, only Interrupt Enable and Error Enable corresponding to Error bit 11 need to be set to 1 by software for the snoop errors to be reported and interrupt to be raised. Interrupt enables and Error enables for Error bits [21:14] in Table 4-10 are not implemented and writes to those bits would be simply ignored. Reads would return zeroes.	R/ W1C

TABLE 4-10 Jbus Common Error Bit Assignments

Error	Bit	Description
Bad_Jbus_Cmd	62	Unrecognized Jbus command received
Rsvd	61:58	Reserved,read-only bits, read as 0
Rsvd	57:22	Reserved,read-only bits, read as 0
Snoop_Error_GR	21	Snoop Error has happened due to Secondary PCI DMA.
Snoop_Error_PCI	20	Snoop Error has happened due to Primary PCI DMA.
Snoop_Error_RD	19	Foreign RD hitting cache line in S , O or M.
Rsvd	18	Reserved,read-only , read as 0
Snoop_Error_RDS	17	Snoop error due to own RDS hitting cache line in S , O or M.
Snoop_Error_RDSA	16	Snoop error due to own RDSA hitting cache line in S , O or M.

TABLE 4-10 Jbus Common Error Bit Assignments

Error	Bit	Description
Snoop_Error_OWN	15	Snoop error due to own OWN hitting cache line in S or M.
Snoop_Error_RDO	14	Snoop error due to own RDO hitting cache line in O or M.
Write_Data_Parity_Err	13	Write Data Cycle Parity error detected by JIO on J_AD bus.
Control_Parity_Err (NEW)	12	Control parity Error Detected by JIO. Only valid if JIO Control and Status Register bit 29 = 1.
Snoop Error(NEW)	11	Coherency state of a line in the I/O cache is wrong.
Jbus Illegal Byte Enable Error(NEW)	10	Jbus Illegal Byte Enable for NCRD transaction
Rsvd	9	Reserved. Has no defined reset state . W1C in Jbus Error Log Register. Corresponding bit in Jbus Error Control Register (bit 9) and Jbus Interrupt Control Register (bit 9) are R/W (Writing a 1 to these bits have no effect as there is no error associated with this bit).
Jbus Illegal Coherency Install State Error (NEW)	8	Illegal data install state of “O”
Reserved	7	Reserved, read-only bits, read as 0
Read_Data_Parity_Err	6	Read Data Cycle Parity error detected by JIO on J_AD bus.
Address_Parity_Err	5	Address Cycle Parity error detected by JIO on J_AD bus.
Jbus_Unmapped_Err	4	This bit gets set when any one of the following conditions is true : -JIO received a Jbus data packet for a DMA read with Read Error indicating an Unmapped Error. -JIO recieves a Write Packet with an address beyond its range. -JIO issues a store to an invalid Jbus port in the system. -JIO issues a read to an invalid Jbus port in the system.
Jbus_Timeout_ Counter_Expired_Err	3	JIO detected Timeout for any read because of Jbus Timer Count Expiration.
Jbus_BusError	2	JIO received a Jbus data packet for a non-cacheable DMA read with Read Error indicating a Bus Error
Jbus_Timeout_Err	1	JIO received a Jbus data packet for a non-cacheable DMA read with Read Error indicating a Timeout
Rsvd.	0	Reserved.

Note – Other design specific errors will probably be added to support debug.

4.2.5 Jbus Parity Control Register

The Parity Control Register controls JIO's detection of Jbus Control and Data parity, and assertion of associated error interrupts. Parity check must be enabled in the Parity Control Register in order for JIO to perform Parity checking.

TABLE 4-11 Parity Control Register(Jbus_Base + 0x41.0020)

Bits	Field	Description	Reset Value	Type
63	DATA_PRTY_EN	Parity Check Enable. Parity is checked on incoming address/data when set to one. Parity is always generated on outgoing address/data.	0	R/W
62	UE_INTEN	Uncorrectable error Interrupt Enable. When set to one a UE Interrupt will be generated if an uncorrectable error is detected.	0	R/W
61	CE_INTEN	Correctable error Interrupt Enable. When set to one a CE Interrupt will be generated if a correctable error is detected.	0	R/W
60:19	Rsvd	Reserved. Read as zero.	0x000 0000 0000	R
18	FCPE	Force Control Parity Error. For diagnostics only.This bit should only be set by software when JIO is configured to generate control parity.	0	R/W
17:14	Rsvd	Reserved	0x0	R
13:10	FDPE[3:0]	Force Data Parity Error. For diagnostics only. 1 bit for each 32 bit data segment. If set to 1, flip data parity on Jbus for that data segment. If set to 0, propagate generated parity on Jbus for that data segment. **Note**: For JIO sending out IDLE transactions on Jbus, only J_ADP bits 0 and 2 can be flipped by setting bits 9 and 12 in this field. Setting bits 11,13 will have no effect on J_ADP bits 1 and 3 respectively for the IDLE transaction.	0	R/W
9:4	Rsvd	Reserved	0x000	R
3:0	Rsvd	Reserved. Read as zero.	0x0	R

Note – The timing of changes to this register when a PIO write is performed is somewhat indeterminate. If software wants to ensure that a change takes effect before proceeding, it should follow the PIO write by a PIO read of this register.

The following table shows how Parity checking, error handling is done in JIO. Further details on how transactions with errors are handled can be found in Chapter 5 "Error Handling and Logging".

TABLE 4-12 Data Parity Error Reporting

DATA_PRTY_EN	JE_INTEN (Jbus Error Interrupt Enable)	Description
0	X	No Data Parity Error checking and reporting, every Jbus transaction proceed as if there is no Data Parity error.
1	0	Data Parity checking is done. If an error is detected, error is logged in Jbus Error Log register but no interrupt is generated. Software should clear error status before enabling interrupt.
1	1	Data Parity checking is done. If an error is detected, error is logged in Jbus Error Log register and JIO generates an interrupt.

TABLE 4-13 Control Parity Error Reporting

CNTL_PRTY_EN (TOM Control & Status Reg, bit 29)	JE_INTEN (Jbus Error InterruptEnable)	Description
0	X	No Control Parity Error checking and reporting, every Jbus transaction proceeds as if there is no Control Parity error.
1	0	Control Parity checking is done. If an error is detected, error is logged in Jbus Error Log register but no interrupt is generated. Software should clear error status before enabling interrupt.
1	1	Control Parity checking is done. If an error is detected, error is logged in Jbus Error Log register and JIO generates an interrupt.

4.2.6 Correctable and Uncorrectable Error Asynchronous Fault Status Registers

These registers (CE AFSR and UE AFSR) log correctable and uncorrectable ECC errors detected by the memory controllers and propagated on Jbus J_adtype. ECC errors may occur on DVMA reads and partial DVMA writes which incur a read-modify-write operation.

“Primary” errors correspond to the first occurrence of a correctable ECC error logged into the CE AFSR and to the first occurrence of an uncorrectable ECC error logged into the UE AFSR. “Secondary” errors corresponds to errors which occur when a primary error is already logged and has not been cleared. This implies that a secondary error bit is set if one of the primary error bit is already set.

The order which defines primary and secondary error is the Jbus data bus order which is not guaranteed to be the same order as the request. It should also be noted that UE and CE errors are looked at only for data which is brought on chip, not necessarily all data transferred on the Jbus data bus.

Secondary errors are cumulative, this means that more than one secondary error bit can be set if two or more errors occurred on different ECC words while a primary error was already logged. It also means that in the case where only one secondary error bit is set, more than one uncorrectable error may have occurred. It should be noted that only a single primary error bit can be set.

A secondary error is simply logged and does not generate an Interrupt.The expectation is that the Interrupt due to the Primary Error would come and clear the secondary error bit along with the primary bit. But in some corner case if that does not happen and the secondary error does not get cleared, the next time an error occurs, it is logged as a primary error and the interrupt due to that clears the pending secondary error bit.

The Correctable and Uncorrectable Error Asynchronous Fault Status registers have the following format:

TABLE 4-14 UE/CE Asynchronous Fault Status Registers(Jbus_Base + 0x41.0030/ 0x41.0040)

Bits	Field	Description	Reset Value	Type
63	Rsvd	Reserved	X	R
62	P_DRD	Primary Error (CE or UE) on DVMA access. Set to one when the error is detected.	X	R/W1C
61	Rsvd	Reserved	X	R
60	Rsvd	Reserved	X	R

TABLE 4-14 UE/CE Asynchronous Fault Status Registers(Jbus_Base + 0x41.0030/0x41.0040)

Bits	Field	Description	Reset Value	Type
59	S_DMA	Secondary Error (CE or UE) on DVMA access. Set to one when the error is detected.	X	R/W1C
58	Rsvd	Reserved. Read as 0.	0	R
57:56	Rsvd	Reserved. Read as 0.	0x0	R
55:42	Rsvd	Reserved. Read as 0.	0x0000	R
41:32	Rsvd	Reserved. Read as 0	0x000	R
31:30	QW_OFFSET	Quad Word Offset inside the 64 byte block on which the primary error was detected.	0xX	R
29	Rsvd	Reserved. Read as 0.	0	R
28:24	JPID	JIO's Jbus Port ID (Port ID of the device that initiated the transaction that had faulty data)	0xXX	R
23:22	Rsvd	Reserved. Read as 0.	0x0	R
21:20	Rsvd	Reserved. Read as zero.	0x0	R
19:13	Rsvd	Reserved. Read as zero.	0x0	R
12:9	Rsvd	Reserved. Read as zero.	0x0	R
8:0	Rsvd	Reserved. Read as zero.	0x0	R

4.2.7 Correctable and Uncorrectable Error Asynchronous Fault Address Registers

These registers (CE AFAR and UE AFAR) log the physical address of the data on which a primary correctable or uncorrectable error occurred.

TABLE 4-15 UE/CE Asynchronous Fault Address Register(Jbus_Base + 0x41.0038/0x41.0048)

Bits	Field	Description	Reset Value	Type
63:44	Rsvd	Reserved. Read as zero	0	R
43	Rsvd	Reserved. Read as zero	0	R
42:4	Address	PA[42:4].	0xXXXXXXXXXX	R
3:0	Rsvd	Reserved. Read as zero	0	R

4.2.8 Jbus Energy Star Control Register

TABLE 4-16 Jbus Energy Star Control Register(Jbus_Base + 0x41.0050)

Bits	Field	Description	Reset Value	Type
63:6	Rsvd	Reserved. Read as zero.	Power-On reset: 0. Preserves its value across Soft reset.	R
5	1/32 Speed	Operate Jbus and interface logic at 1/32nd input frequency.	Power-On reset: 0. Preserves its value across Soft reset.	R/W
4:2	Rsvd	Reserved. Read as zero.	Power-On reset: 0. Preserves its value across Soft reset.	R
1	1/2 Speed	Operate Jbus and interface logic at 1/2 the frequency of the input clock.	Power-On reset: 0. Preserves its value across Soft reset.	R/W
0	Full Speed	Operate Jbus and interface logic at the same frequency as the input clock.	Power-On reset: 1. Preserves its value across Soft reset.	R/W

One and only one bit shall be set at a time.

4.2.9 Jbus Change Initiation Control Register

TABLE 4-17 Jbus Change Initiation Control Register(Jbus_Base + 0x41.0058)

Bits	Field	Description	Reset Value	Type
63:5	Rsvd	Reserved. Read as zero.	-	R
4:3	Chng_Init[1:0]	At reset equals 2'b00. Should be set by software only in JIO with JPID[2:0] = 6. Software writes 2'b10 to initiate the Change sequence. A write of 2'b11 is a software error. JIO then initiates Change transaction on Jbus and changes this field to 2'b11. Software then comes and reads 2'b11 and clears it to 2'b00.	Power-On reset: 0x0. Preserves its value across Soft reset.	R/W
2:0	Chng_Delay[2:0]	J_CHANGE_L propagation cycles in the system to all valid Jbus agents. Valid values are 0,1, 2,3,4,5. Set by OBP. This represents the delay in Jbus clock cycles in using the J_CHNG_L internally to match the delay that all other valid Jbus agents see in getting J_CHNG_L. The recommended Programming Value is 0x0 for Enchilada (no repeater) and 0x2 for Chalupa (repeater has 2 clock cycle delay).	Power-On reset: 0x0. Preserves its value across Soft reset.	R/W

4.2.10 Jbus DTag Diagnostic Registers

The DTags are the dual tags that are used to maintain consistency on the contents of the JIO PCI I/O cache.The PCI I/O cache is used for any partial DMA writes (quantities of less than 64 bytes) to cacheable addresses. When an entry is allocated inside the PCI I/O cache, JIO assumes ownership of that line, and Jbus addresses snoop the DTags for a possible hit and the current state of the line.

The DTags are made accessible for diagnostic purposes. It should be noted that read latency is large when compared to the snoop delay. If Jbus transactions are changing the values, the value read may differ significantly from its value when the NCRD was on the bus.

TABLE 4-18 DTag Diagnostic Register(Jbus_Base + {0x41.2000 - 0x41.2070})

Bits	Fields	Description	Reset Value	Type
63:59	Rsvd	Reserved. Read as zero.	0	R
58:56	S	Cache line State Valid encodings are : I = 3'b000 S = 3'b001 O = 3'b101 M = 3'b111	0xX	R/W
55:42	Rsvd	Reserved. Read as zero.	0	R
41:6	PA[41:6]	Physical Address	0xXX XXXX XXXX	R/W
5:0	Rsvd	Reserved. Read as zero.	0	R

S identifies the Cache line state of the entry.

Note – There are 8 separate DTag diagnostic registers (one for each entry of the PCI I/O cache) aligned on 16 byte boundaries although they are only 8 byte long.

Note – It is illegal for Software to program the S field in DTag Diagnostic Registers as M (3'b111) while Disable_M_State bit (bit 53 of JIO Control and Status Register(Jbus_Base + 0x41.0000)) has been programmed to 1'b1. Such a programming will not be blocked by JIO but will lead to data corruption issues in the system.

4.2.11 Jbus CTag Diagnostic Registers

The CTags are the dual tags that are used to maintain consistency on the contents of

TABLE 4-19 CTag Diagnostic Register(Jbus_Base + {0x41.3000 - 0x41.3070})

Bits	Fields	Description	Reset Value	Type
63:59	Rsvd	Reserved. Read as zero.	0	R
58:56	S	Cache line State Valid encodings are: I = 3'b000 S = 3'b001 O = 3'b101 M = 3'b111	0xX	R/W
55:42	Rsvd	Reserved. Read as zero.	0	R
41:6	PA[41:6]	Physical Address	0xXX XXXX XXXX	R/W
5:0	Rsvd	Reserved. Read as zero.	0	R

the JIO graphics (GR) I/O cache.The GR I/O cache is used for any partial DMA writes (quantities of less than 64 bytes) to cacheable addresses. When an entry is allocated inside the GR I/O cache, JIO assumes ownership of that line, and Jbus addresses snoop the Ctags for a possible hit and the current state of the line.

The CTags are made accessible for diagnostic purposes. It should be noted that read latency is large when compared to the snoop delay. If Jbus transactions are changing the values, the value read may differ significantly from its value when the NCRD was on the bus.

S identifies the Cache line state of the entry.

Note – There are 8 separate CTag diagnostic registers (one for each entry of the GR I/O cache) aligned on 16 byte boundaries although they are only 8 byte long.

Note – It is illegal for Software to program the S field in CTag Diagnostic Registers as M (3'b111) while Disable_M_State bit (bit 53 of JIO Control and Status Register(Jbus_Base + 0x41.0000)) has been programmed to 1'b1. Such a programming will not be blocked by JIO but will lead to data corruption issues in the system.

4.2.12 Jbus Performance Control Register

JIO includes two 32-bit counters that can be used to gather statistics on Jbus related hardware events.

The type of events that are counted by the performance counters is specified through the Jbus Performance Control register. The format is:

TABLE 4-20 Jbus Performance Control Register(Jbus_Base + 0x41.7000)

Bits	Fields	Description	Reset Value	Type
63:16	Rsvd	Reserved. Read as zero.	0	R
15:11	Cnt1 Select	Event selection for counter 1	0	R/W
10:9	Rsvd	Reserved. Read as zero.	0	R
8:4	Cnt0 Select	Event selection for counter 0	0	R/W
3:0	Rsvd	Reserved. Read as zero.	0	R

The list of events with their selection codes is:

TABLE 4-21 Jbus Events Selection codes:

Code	Jbus Event
0x00	Counting Disabled.
0x01	Jbus Bus cycles
0x02	Cycles AOK is asserted by this JIO (flow control).
0x03	Jbus Coherent transactions (i.e. transaction in memory space)
0x04	Reserved
0x05	JIO's own coherent transactions.
0x06	Snoop hits in the I/O caches.
0x07	All Jbus NC transactions (i.e. all NCRD, NCWR, NCBRD, NCBWR transactions on Jbus, including ones generated by JIO).
0x08	Foreign PIO hits (i.e. NC transactions in the NC space mapped to this JIO).
0x09	Number of writebacks issued from JIO
0x0a	Interrupts issued.
0x0b	Number of WRI, WRIS issued from JIO
0x0c	PIO accesses to JIO's internal registers
0x0d	UPA accesses

TABLE 4-21 Jbus Events Selection codes:

Code	Jbus Event
0x0e	PCI A accesses
0x0f	PCI B accesses
0x10 - 0x1F	Reserved

Note – only 0x0 through 0x8 are implemented. The rest are requests and are being investigated.

4.2.13 Jbus Performance Counters Register

The performance counters are wrap around counters. They are 32 bit wide so with a 150 Mhz Jbus clock they wrap in about 28.6 seconds when counting bus cycles. Note that since the only legal access size is 8 bytes, both count fields must be updated (e.g. set to 0) together.

TABLE 4-22 Jbus Performance Counters Register(Jbus_Base + 0x41.7008)

Bits	Fields	Description	Reset Value	Type
63:32	Cnt1	Counter 1	0	R/W
31:0	Cnt10	Counter 0	0	R/W

4.2.14 UPA Reset Control Register

JIO needs to have the following output pins which will drive the UPA GFX device:

(i) **UPA_RESET_L**: Asserted Low by JIO to put UPA graphics in reset. This signal needs to be software controlled. Software will take the UPA device in and out of reset. As soon as power is applied, this signal needs to be asserted and kept asserted until software deasserts it, and later on during E* mode software will decide when to assert, and when to deassert the signal.

(ii) **UPA_POK**: After applying power (prior to generation of resets) this signal needs to be deasserted by JIO and kept deasserted. Software will control when to deassert the signal during E* mode, and when to assert in normal mode.

This register controls the Reset and PowerOK signals that are issued by JIO to the UPA slot. Whenever JIO is reset (by any reset condition), bits[1:0] are cleared. This means the UPA64S device and JIO's UPA logic are held in reset as longs as POST/

OBP has not written to this register. The PowerOK signal must be set to one for the UPA64S devices' voltage regulator to come up and the PLLs to lock before they are released from reset. Also bit 2 gets reset to 1, thereby enabling clock to UPA logic in JIO.

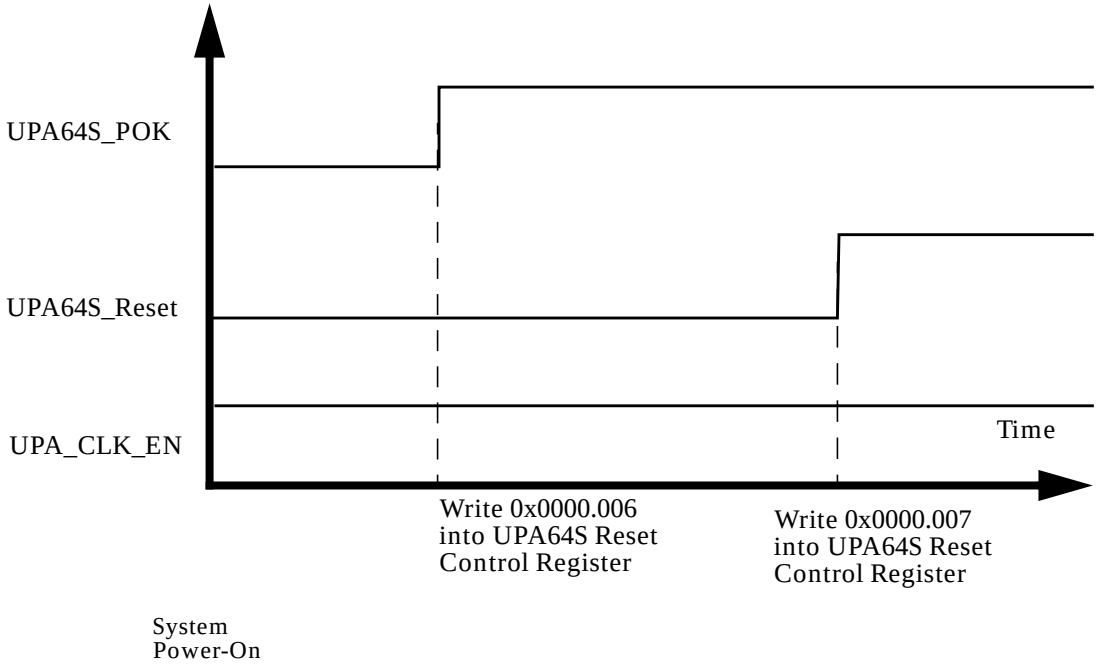
The format for the UPA64S Reset Control register is:

TABLE 4-23 UPA64S Reset Control Register(Jbus_Base + 0x41.7020)

Bits	Fields	Description	Reset Value	Type
63:3	Rsvd	Reserved. Read as zero.	0	R
2	UPA_CLK_EN	This bit enables / disables clock to JIO's internal UPA logic. 0 : Disable UPA clock 1 : Enable UPA clock	1	R/W
1	UPA64S_POK	Power OK. This bit is used to control the Power_OK signal which goes to UPA slot. This signal is active when this bit set to one	0	R/W
0	UPA64S_RST_L	Resets the UPA device. The reset is active as long as this bit is set to zero	0	R/W

The following diagram shows how the UPA64S devices must be taken out of reset.

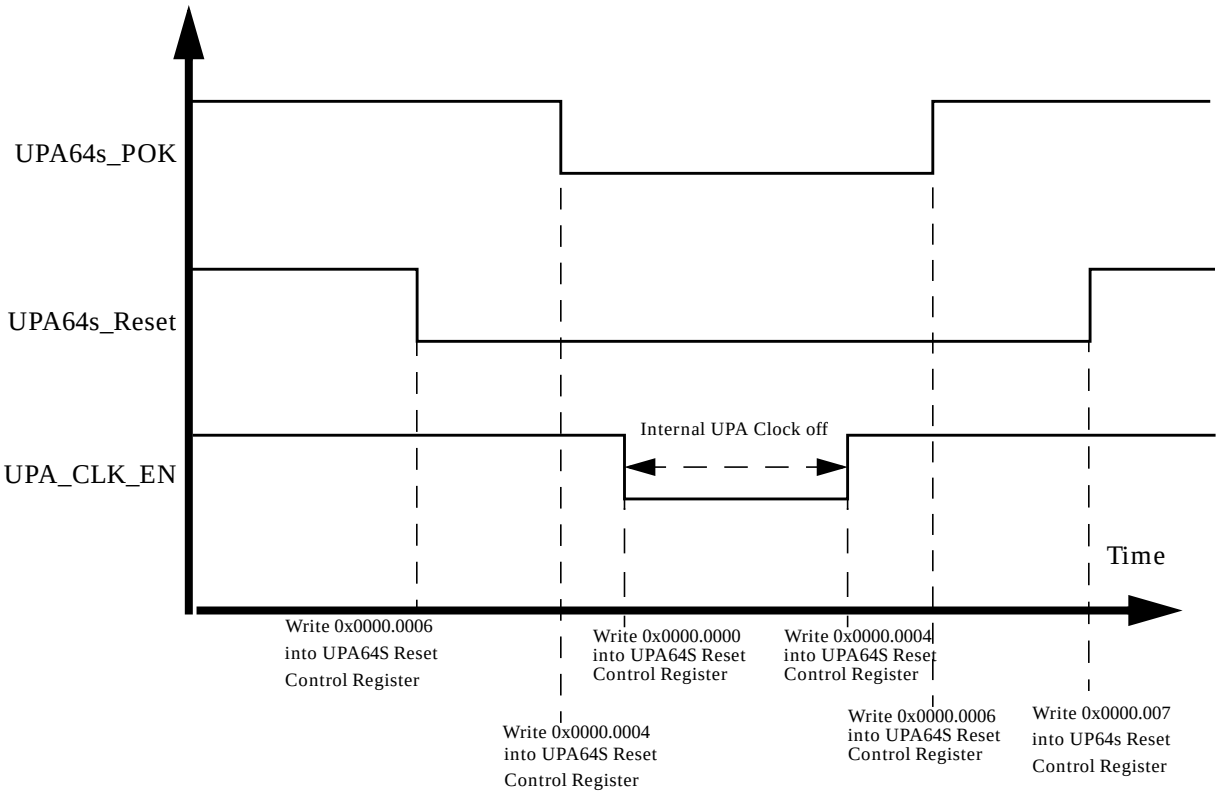
FIGURE 4-2 Software Reset of UPA64S Devices



This register is also used to control powering down the UPS64S device in Energy Star (E*) mode. UPA64S does not provide any specific hook to run at a lower frequency, hence it will be shut down in E* mode by software. Software, in order to put the UPA64S device in low power mode, has to hold the device in reset and deassert the POK signal. By deasserting the POK signal the UPA64S devices' PLLs will be suspended. This will minimize the on board (UPA64S device) dynamic power. The sequencing of the UPA64S reset and POK signals are performed through the UPA64 Interface Configuration Register.Also while both POK and Reset are held low, software can disable JIO's internal UPA clock off as shown and then re-enable it, before turning POK back on.This would save power in the UPA block in JIO during Estar Mode.

The following diagram shows how the graphics board is sequenced in and out of E* mode

FIGURE 4-3 E* Transition for UPA64S



Note – Before causing software reset or Estar of UPA, software should make sure that all prior UPA traffic has made it out of JIO properly, otherwise some PIOs may be lost in JIO. Software should do a pio load / membar #sync to make sure that all prior UPA reads and writes have been flushed out of JIO to both UPA devices, before it causes reset or Estar of UPA.

4.2.15 GPIO Registers

JIO has four General-Purpose Input/Output (GPIO) pins. These pins are used as two pseudo-I2C buses in the system. Software will mimic I2C protocol through “bit-banging” the GPIO registers. However, these pins can be used for other purposes, as seen fit. Upon reset, all pins come up as inputs.

The pins are bidirectional, and have ability to drive LVTTL (3.3V). In addition, they are 5V tolerant on the input.

TABLE 4-24 GPIO 0 Data Register(Jbus_Base + 0x46.0000)

Bits	Fields	Description	Reset Value	Type
7:1	Rsvd	Reserved. Read as zero.	0	R
0	GPIO Pin 0	GPIO Pin 0 Data	0	R/W

TABLE 4-25 GPIO 1 Data Register(Jbus_Base + 0x46.0001)

Bits	Fields	Description	Reset Value	Type
7:1	Rsvd	Reserved. Read as zero.	0	R
0	GPIO Pin 1	GPIO Pin 1 Data	0	R/W

TABLE 4-26 GPIO 2 Data Register(Jbus_Base + 0x46.2000)

Bits	Fields	Description	Reset Value	Type
7:1	Rsvd	Reserved. Read as zero.	0	R
0	GPIO Pin 2	GPIO Pin 2 Data	0	R/W

TABLE 4-27 GPIO 3 Data Register(Jbus_Base + 0x0x46.2001)

Bits	Fields	Description	Reset Value	Type
7:1	Rsvd	Reserved. Read as zero.	0	R
0	GPIO Pin 3	GPIO Pin 3 Data	0	R/W

TABLE 4-28 GPIO Data Register(Jbus_Base + 0x46.4000)

Bits	Fields	Description	Reset Value	Type
63:4	reserved	Rsvd. Read as zeros	0	R
3	GPIO Pin 3	GPIO pin 3 Data	0	R/W
2	GPIO Pin 2	GPIO pin 2 Data	0	R/W
1	GPIO Pin 1	GPIO pin 1 Data	0	R/W
0	GPIO Pin 0	GPIO Pin 0 Data	0	R/W

Note here that bits [3:0] are aliased to by the GPIO pin data bits from the individual GPIO data registers.

TABLE 4-29 GPIO Control Register(Jbus_Base + 0x46.4008)

Bits	Fields	Description	Reset Value	Type
63:4	reserved	Rsvd. Read as zeros.	0	R
3	GPIO 3 direction	Direction of GPIO Pin 3: 0: Input 1: Output	0	R/W
2	GPIO 2 direction	Direction of GPIO Pin 2: 0: Input 1: Output	0	R/W
1	GPIO 1 direction	Direction of GPIO Pin 1: 0: Input 1: Output	0	R/W
0	GPIO 0 direction	Direction of GPIO Pin 0: 0: Input 1: Output	0	R/W

4.3 UPA Leaf Interface

TABLE 4-30 UPA Register Offsets

Register	Offset	Access Size
UPA Slot0 Configuration Register	0x48.0000	8 bytes
UPA Slot1 Configuration Register	0x48.0008	8 bytes
UPA Interface Configuration Register	0x48.0010	8 bytes
UPA Energy Star Control Register	0x48.0018	8 bytes

4.3.1 UPA64S Configuration Register

The UPA64S Slot Configuration register is provided for compliance with the UPA specification. The reset bit has moved to the UPA64S Interface Configuration register.This register is used to set the slave queue size.

The format for Slot Configuration registers is:

TABLE 4-31 UPA64S Configuration Register(Jbus_Base + 0x48.0000 / 0x48.0008)

Bits	Fields	Description	Reset Value	Type
63	Slot Empty	This field is used to detect the presence of a board in the slot. It is set to one if the slot is empty and cleared by hardware to zero if a board is detected. If a slot is empty the corresponding Offset Base and Mask registers are considered invalid.	-	R
62:43	Rsvd	Reserved, Read as zero	0	R
42:38	Pending Total Transactions	Multiple outstanding transactions supported for graphics cards. 00000: Illegal 00001-10000 One through sixteen outstanding transactions 10001-11111 Illegal	0x1	R / W
37:33	Pending Writes	Multiple outstanding writes supported for graphics cards. 00000: Illegal 00001-10000 One to sixteen writes 10001-11111: Illegal	0x1	R / W

TABLE 4-31 UPA64S Configuration Register(Jbus_Base + 0x48.0000 / 0x48.0008)

Bits	Fields	Description	Reset Value	Type
32:30	Pending Reads	Multiple outstanding reads supported for graphics card. 000 : Illegal 001 : OneRead 010 : two reads 011: Three reads 100: Four Reads 101 - 111 : Illegal	0x1	R / W
29:28	Reserved	0x00	0x0	R
27:24	SPRQS	Slave P_Req queue size. JIO does not use this parameter.	0x1	R / W
23:18	SPDQS	Slave Port Data queue size. JIO does not use this parameter.	0x4	R
17:16	Rsvd	Reserved. Read as zero.	0	R
15	SQUEN	Slave Queues Enable. This bit must be set to one when writing this register. It is always read as zero. JIO does not use this bit.	0 (R0)	W
14	Reserved.	Reserved. Read as zero.	0	R
13:0	Rsvd	Reserved. Read as zero.	0	R

I / O space address decoding (see Jbus Offset Base and Mask Registers) is qualified by the presence of a device in the UPA slot. That is, when Slot Empty is set the corresponding address decoder is disabled and MappedOut is not asserted for the address region. This prevents transactions from timing out because no device is present to respond.

4.3.2 UPA64S Interface Configuration Register

The format for the UPA64S Interface Configuration register (UPA64S_ICR) is:

TABLE 4-32 UPA64S Interface Configuration Register(Jbus_Base + 0x48.0010)

Bits	Fields	Description	Reset Value	Type
63:11	Rsvd	Reserved. Read as zero.	0	R
10:5	UPA_PLL_ TUNE[5:0]	UPA PLL Tune Bits. The Tune bits are used to modify the PLL loop parameters to compensate for high levels of noise or input jitter.This TUNE bit programming capability provide the flexibility for the system engineer to optimize the PLL stability and jitter. Normally, the PLL must be reset when this field is changed by software. The only exception to this (and the recommended way of changing this field without causing a PLL reset) is to change the tune bits 1 LSB at a time, with a hold of 1 microsecond at each step. For instance, to move from TUNE(5:0)='010010' to TUNE(5:0)='010101' would require the following sequence, and a hold of 1us at each point. Initial TUNE(5:0)='010010' Mid-1 TUNE(5:0)='010011' Mid-2 TUNE(5:0)='010100' Final TUNE(5:0)='010101'	0x12	R/W
4	UPA64s_Stall_ Store	When set to 1 inhibits JIO's UPA logic from sending stores to an UPA64 device while there is an outstanding read to an UPA64S device.	0	R/W

TABLE 4-32 UPA64S Interface Configuration Register(Jbus_Base + 0x48.0010)

Bits	Fields	Description	Reset Value	Type
3	UPA64s_Sreply ReadData_ Latency	Sreply to data valid latency control. Adds one clock latency to read of UPA graphics data internally.Latency is transparent to system board. 0: No added latency. 1: One clock extra latency. Note: For each value of UPA64S_Sreply_ReadData_ Latency, and motherboard delay on the UPA read clock, software should read a known location from the UPA64S device, and complement the UPA64S_ Toggle_Read_Ptr, if the correct value is not read. If the correct value is still not read, the motherboard delay on the UPA read clock may be too large or too small. The exact min-max spec depends on the final analysis of JIO timing.	1	R/W
2	UPA64S_Toggle_ Read_Ptr	This bit, when set complements the reset state of the Read pointer for the 2 entry UPA64S read data fifo, so it has the opposite relationship to the write pointer.	0	R/W
1:0	Rsvd.	Reserved. Read as zero.	0	R

4.4 Scratch Pad Register

OBP needs this scratch pad register. It is a continuous 512 Byte block (looks like a RAM to software). Address is accessible directly from CPU. Software does not require to setup any registers to access the 512 bytes.

TABLE 4-33 Scratch Pad Register (PCI-A_CSRBase +{0x0.0000.2040 - 0x0.0000.2238})

Register	Address	Access Size
Scratch Pad Register	PCI-A_CSRBase +0x0.0000.2040 - 0x0.0000.2238	Any, provided within 8 byte boundary

Note – For both reads and writes to the scratchpad, physical address bits [2:0] do not get looked at by JIO. For a write, software can access any number of bytes (1/2/4) with proper byte enables within a 8 byte boundary. For a read, JIO would return the content of the entire 8 byte register (ignoring address bits [2:0])and CPU aligns the data w.r.t the byte enables. This is true for all CSR reads to JIO. However , software should not do a write or a read with byte enables turned on that cross a 8 byte boundary.In that case, depending on PA[3], the lower or upper 8 byte segment will be read/written. The other half will be dumped. Then PA[3] can be toggled and a new access send to access the other half.

4.5 PCI Leaf Interface

Note – JIO contains two nearly identical copies of a PCI Leaf Block. The programmer's model for both copies is identical. Register addresses are shown with respect to the base address PCI_CSRBase, whose real value should be one of the two base addresses PCI-A_CSRBase or PCI-B_CSRBase. Registers which are in PCI Configuration Space are shown with respect to PCI_ConfigBase, which again should be the appropriate PCI-A_ConfigBase or PCI-B_ConfigBase address.

4.5.1 PCI Bus Module

Within a PCI Leaf, the block directly responsible for the PCI transactions and protocol is the PCI Bus Module (PBM), which has a set of control/status registers. Some of these registers control aspects of JIO's PCI operations that are not completely defined, or left implementation dependent by the PCI specification. These PBM registers are placed into the PCI_CSRBase address region. The PBM also has a number of registers in PCI Configuration Space which are defined by the PCI specification.

TABLE 4-34 Offset of PBM Registers

Register	Address	Access Size
PCI Control/Status Register	PCI_CSRBase+0x0.0000.2000	8 bytes
PCI PIO AFSR	PCI_CSRBase+0x0.0000.2010	8 bytes
PCI PIO AFAR	PCI_CSRBase+0x0.0000.2018	8 bytes

TABLE 4-34 Offset of PBM Registers

Register	Address	Access Size
PCI Diagnostic Register	PCI_CSRBase+0x0.0000.2020	8 bytes
PCI Energy Star Register	PCI_CSRBase+0x0.0000.2028	8 bytes
PCI Target Retry Limit (NEW)	PCI_CSRBase+0x0.0000.2030	8 bytes
PCI Target Latency Timer (NEW)	PCI_CSRBase+0x0.0000.2038	8 bytes
Scratch Pad Register (NEW, for PCI-A_CSRBase only)	PCI-A_CSRBase + 0x0.0000.2040 - 0x0.0000.2238	8 bytes
Interrupt Routing Register (NEW, for PCI-A_CSRBase only)	PCI-A_CSRBase + 0x0.0000.2240	8 bytes
PCI Target Address Space Register (NEW)	PCI_CSRBase+0x0.0000.2490 and 0x54 in PCI Config Space	8 bytes
PCI Target Error VA Log Register (NEW)	PCI_CSRBase+0x0.0000.2498	8 bytes

4.5.1.1 PCI Control/Status Register

.

TABLE 4-35 PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)

Field	Bits	Reset	Description	R/W
Rsvd	63	0	Reserved, read as 0	R
PCI_DTO_ERR	62	0	PCI Discard Timeout Error Set to 1 when a discard timeout is detected for any read issued by PCI.	R/W1C
DTO_INT_EN	61	0	Discard Timeout interrupt Enable 0 = Discard timeouts will not cause a PCI interrupt to be issued . 1 = Discard timeouts will cause a PCI interrupt to be issued.	R/W

TABLE 4-35 PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)

Field	Bits	Reset	Description	R/W
ARB_PRIORITY[8:0]	60:52	0x0	This 9 bit field allows software to set any particular request to low priority by writing a 1 to the corresponding bit. Bit 60 corresponds to JIO and bits 59:52 correspond to External PCI master slots 7:0. The PCI arbiter round robins between the low priority requests, and their winner is treated as a peer when arbitrating amongst the high priority requests. If all the priority bits are set or cleared, all requests are of equal priority. Note that if software wants to have a two tier arbitration with one or more low priority requests, it has to minimally write a 1 to bit 59 to make the PCI master connected to slot 7 a low priority. It can then selectively make any other slot low priority by writing a 1. The two tier arbitration will not work unless slot 7 has been assigned low priority.	R/W
Rsvd	51	0	Reserved, read as 0	R
Rsvd	50:39	0	Reserved, read as 0	R
PCI_TTO_ERR	38	0	PCI TRDY# Timeout error Set to 1 when a TRDY# timeout is detected during any PIO by JIO	R/W1C
PCI_RTRY_ERR	37	0	PCI Excessive Retry error Set to 1 if the maximum retries are exceeded during any PIO.	R/W1C
PCI_MMU_ERR	36	0	PCI IOMMU Error. Set to 1 if an IOMMU Error is detected for any DMA Note that this bit gets set at the same time ERR bit (bit 24) in IOMMU Control Register gets set. But it gets cleared separately (than the ERR bit) by software.	R/W1C
Rsvd	35	0	Reserved, read as 0	R
PCI_SERR	34	0	Set when SERR# signal is sampled asserted on the PCI bus	R/W1C
Reserved	33	-	Reserved, Read as 0.	R
Reserved	32:31	0	Reserved, Read as 0.	R

TABLE 4-35 PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)

Field	Bits	Reset	Description	R/W
PEN_RD_MLTPL	30	0	Enable prefetch in PBM if PCI command is “Memory Read Multiple” . 0: Disable 1: Enable. prefetch next line if PCI Target Read Data fifo is empty and no end of transfer yet for the current transaction.	R/W
PEN_RD_ONE	29	0	Enable prefetch in PBM if PCI command is “Memory Read” . 0: Disable 1: Enable. Prefetch next line if PCI Target Read Data fifo is empty and no end of transfer yet for the current transaction.	R/W
PEN_RD_LINE	28	0	Enable prefetch in PBM if PCI command is “Memory Read Line” . 0: Disable 1: Enable. Prefetch next line if PCI Target Read Data fifo is empty and no end of transfer yet for the current transaction.	R/W
FRC_TRGT_ABRT	27	0	When set to 1, JIO signals a target abort on the PCI bus if the bus is busy and JIO is the target.	R/W
FRC_TRGT_RTRY	26	0	When set to 1, JIO signals a retry on the PCI bus if the bus is busy and JIO is the target.	R/W
PTO	25:24	0	PCI Timeout interval. See TABLE 4-37 for definition.	R/W
TRGT_RW_STL_WT	23:21	0x0	Count after which JIO will issue a retry due to the DMA Target Write fifo being full (on a DMA write to JIO) or DMA Target Read Fifo being empty (on a DMA read from JIO).	R/W
Reserved	20	0	Reserved, read as 0.	R
MMU_INT_EN	19	0	IOMMU Error Interrupt Enable 0 = IOMMU errors will not cause a PCI error to be issued 1 = IOMMU errors will cause a PCI error to be issued Note that setting this bit qualifies any IOMMU error to be qualified as a PCI Error which would then assert an interrupt only if ERRINT_EN bit in this register is set.	R/W
Reserved	18	0	Reserved, read as 0.	R
ERRINT_EN	17	0	Enable PCI error interrupt. 0 = PCI error interrupt disabled 1 = PCI error interrupt enabled	R/W

TABLE 4-35 PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)

Field	Bits	Reset	Description	R/W
ARB_PARK	16	1	0 = the default driver for the PCI bus is always JIO , i.e. the grant returns to JIO after every transaction if there are no other requestors and the PCI bus is idle. (this setting limits JIO's PCI PIO performance , so should not be used when high PIO performance is required. Please see note below table) 1 = PCI bus grant stays asserted to the last requestor while the PCI bus is idle so that device is the default driver. (This has some performance benefits, since there will be lower latency for devices to access the bus if they are doing spurts of transactions, without other devices, or PIO operations from JIO intervening)	R / W
Reserved	15	0	Reserved, read as 0.	R

TABLE 4-35 PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)

Field	Bits	Reset	Description	R/W
PCI_PLL_TUNE[5:0]	14:9	**PCIA** : 66 Mhz : 0x15 33 Mhz : 0x16 **PCIB** : 66 Mhz : 0x15 33 Mhz : 0x16	PCI PLL Tune bits. The reset value for the PCIA and PCIB leaves are dependant on the 33/66Mhz modes of operation of the respective PCI Buses. The Tune bits are used to modify the PLL loop parameters to compensate for high levels of noise or input jitter.This TUNE bit programming capability provide the flexibility for the system engineer to optimize the PLL stability and jitter. Normally, the PLL must be reset when this field is changed by software. The only exception to this (and the recommended way of changing this field without causing a PLL reset) is to change the tune bits 1 LSB at a time, with a hold of 1 microsecond at each step. For instance, to move from TUNE(5:0)='010010' to TUNE(5:0)='010101' would require the following sequence, and a hold of 1us at each point. Initial TUNE(5:0)='010010' Mid-1 TUNE(5:0)='010011' Mid-2 TUNE(5:0)='010100' Final TUNE(5:0)='010101'	R/W
Reserved	8	0	Reserved, read as 0.	R
ARB_EN<7:0>	7:0	0x00	PCI DMA arbitration enable. One independent bit for each potentially supported external master device on the bus. 0 = Bus requests from corresponding PCI device are ignored 1 = Bus requests from corresponding PCI device are honored.	R/W

Note – If ARB_PARK bit is programmed to 1'b0 (JIO gets the grant after every transaction if there are no other requestors and the PCI bus is idle), the maximum data beats JIO can issue in the new bus parking mode are two data beats. If there are more than two data beats to transfer, JIO will break the transaction to multiple two-data-beat transactions. The reason is that, in this mode , to avoid bus contention , grant is only propagated to the PCI master machine in JIO when the PCI bus is IDLE, and the grant is deasserted while the bus is BUSY . So JIO samples the grant after it issues the first data-beat. Since the internal grant is deasserted while BUS is busy, the master machine will send another data-beat and back off the bus. And this happens even if no other PCI agent is requesting the bus .

External/Internal Arbiter Detection

JIO will have two internal arbiters, one on each PCI Leaf. There may be also one external arbiter for each leaf. JIO can support any combination of internal and external arbiters for its two PCI leaves in any system configuration. Depending on the mode in which the Graphics side of JIO is configured in the current system configuration (UPA or PCI), it is possible to statically figure out during reset, whether the internal or external arbiter is enabled on each PCI leaf by looking at

PCIA req1 and PCIB req 1.The table below shows how JIO's internal PCI logic will detect the presence of internal/external PCI arbiter on each side in any system configuration during reset.

TABLE 4-36 External / Internal Arbiter Detection

PCIA req1	PCIB req1	g_upa_ pci_sel	External/Internal Arbiter Detection
1	x	x	a) Primary (A) internal arbiter enabled, by observing a one on PCIA request 1 (THIS WILL REQUIRE THE MOTHER BOARD TO TIE THIS PIN TO A PULLUP); b) Primary side external arbiter is disabled. c) Since the internal arbiter for the A side is enabled, the grant to the internal PCI master (with request routed to PCIA req 8) will be driven active after reset to "park" the JIO master, and have the JIO drive the otherwise undriven primary side AD and CBE buses.
0	x	x	a) The primary side internal arbiter is disabled by observing a low on PCIA request 1 (THIS WILL REQUIRE THE MOTHER BOARD TO TIE THIS PIN LOW AND NOT ROUTE THIS REQUEST PIN TO THE CONNNECTOR) b) Primary side external arbiter is enabled. c) The primary PCI master's PCI request is routed to PCIA req0, and after reset, the PCI master will drive it high.
x	1	0	a) With the PCIB req1 observed high in reset (THIS WILL REQUIRE THE MOTHER BOARD TO TIE THIS PIN TO A PULL UP), the secondary bus internal arbiter is enabled. b) Secondary side external arbiter is disabled. c) The secondary arbiter will park the secondary side JIO master via a internal grant to the PCI master.Secondary side's req gets routed to the internal arbiter as PCIB req 8.
x	0	0	a) The secondary or B side PCI arbiter is disabled by observing a low on PCIB request 1(THIS WILL REQUIRE THE MOTHER BOARD TO TIE THIS PIN LOW AND NOT ROUTE THIS REQUEST PIN TO THE CONNNECTOR) b) Secondary side external PCI arbiter is enabled. c) The secondary or B side PCI master request is routed to PCIB req 0, and after reset the secondary master will drive it high.

PCI Timeouts

JIO is capable of detecting several different types of timeouts as error conditions on the PCI bus:

- TRDY# Timeout (TTO) - When JIO is a PCI master and a slave that has already claimed the cycle with DEVSEL# takes too many cycles to assert TRDY#, JIO will terminate the transaction and will signal this error.
- Max retries exceeded (RTRY) - When JIO is a PCI master doing a PIO read or write, and the target device terminates the transaction with a retry more than the specified number of times in a row, JIO will abort the transaction and signal this error.
- Discard Timeout (DTO) - If a PCI master has not issued a PCI Delayed Read request within a certain number of PCI cycles, JIO discards any data it may have for the read transaction, and signals this error.

The durations of each of these timeouts are controlled globally by the PTO field in the PCI Control/Status Register, which has the following definition:

TABLE 4-37 PCI Timeout Intervals

PTO<1:0>	Maximum Retries (in # retries)	TRDY# Timeout (in # bus cycles)	Discard Timeout (in # bus cycles)	Intended Purpose
00	∞ (disabled)	∞ (disabled)	∞ (disabled)	Reset value. Used for debug/boot.
01	2^14	2^15	2^15	Bringup/lab testing
10	2^11	2^12	2^12	Bringup/lab testing
11	2^14	2^15	2^8	Standard value (for production systems)

Note – The global disable provided by PTO=0x0 is in addition to individual disables that may exist (e.g. for max retries and TTO error).

Note – At reset, PTO = 00 (all disabled) . Then software should program it to 2'b11 so that we have DTO after 2^8 cycles.If software wants to disable TTO and RTO, software can make DIS_TTO = 1 and DIS_RETRY =1 in PCI diagnostic register.

Note – If there is a DTO causing event on the PCI bus with JIO as the target (DMA read request doesn't return data), and the PTO is 2'b00, no DTO gets logged, and JIO will hang, because it requires the DTO event to clear the pending read. Thus PTO==2'b00 (disabled) should never be used in production systems.

4.5.1.2 PCI PIO Asynchronous Fault Status / Address Registers

PCI AFSR / AFAR record error information related to PIO writes to PCI slave devices. Only asynchronous errors reported through interrupt are recorded in these registers. Asynchronous errors include any PIO write access terminated by Master Abort, Target Abort, or excessive retries, as well as any PIO write during which a data parity error was signaled on the PCI bus. Although status bits for Master Abort, Target Abort and Parity Error exist in the PCI Configuration Registers for each PBM, they are duplicated here to provide the additional functionality of identifying which error occurred first in the case of multiple errors, and associating an address with that error.

Two sets of status bits are defined in this register. Bits <63:59> are the primary error status and bits <57:53> are the secondary error status. One and only one of the primary error status can be set at any time. Primary error status can be set only when:

- None of the primary error condition exists prior to this error OR
- New error detected at the same time software is clearing the primary error.

Secondary bits are set whenever a primary bit is set (one and only one primary bit can be set at a time). The secondary bits are cumulative and always indicate that information has been lost as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits <47:37> of AFSR logs address and status of the primary PCI PIO error. Further PCI PIO error will not be logged into these bits until software clears the primary error, which makes the AFAR and part of the AFSR available to log new error. An interrupt is generated, if enabled, whenever the AFAR logs the new error address.

TABLE 4-38 PCI PIO AFSR(PCI_CSRBase+0x0.0000.2010)

Field	Bits	Reset	Description	R/W
P_MA	63	0	Set if primary error detected is Master Abort	R/W1C
P_TA	62	0	Set if primary error detected is Target Abort	R/W1C
P_RTRY	61	0	Set if primary error detected is excessive retries	R/W1C
P_PERR	60	0	Set if primary error detected is data parity error	R/W1C
P_TTO	59	0	Set if primary error detected is TRDY# timeout	R/W1C
Rsvd	58	0	Reserved. Read as zero.	R
S_MA	57	0	Set if secondary error detected is Master Abort	R/W1C
S_TA	56	0	Set if secondary error detected is Target Abort	R/W1C
S_RTRY	55	0	Set if secondary error detected is excessive retries	R/W1C

TABLE 4-38 PCI PIO AFSR(PCI_CSRBase+0x0.0000.2010)

Field	Bits	Reset	Description	R/W
S_PERR	54	0	Set if secondary error detected is data parity error	R/W1C
S_TTO	53	0	Set if secondary error detected is TRDY# timeout	R/W1C
Rsvd	52	0	Reserved.	R
Reserved	51:40	0	Reserved, read as 0	R
MASK	39:32	0	PCI Byte Enables <7:0 > for transfer in Error	R
BLK	31	0	Set to 1 if failed primary transfer was a block read or write from Jbus	R
ConfigSpace	30	0	Set to 1 if failed primary transaction was to PCI Configuration Space.	R
MemorySpace	29	0	Set to 1 if failed primary transaction was to PCI Memory Space	R
IOSpace	28	0	Set to 1 if failed primary transaction was to PCI I/O Space	R
Reserved	27:0	0	Reserved, read as 0	R

Note – The PCI PIO AFSR/AFAR values get cleared on power-on and soft resets, as Solaris does not look at these registers after reset.

Note – Bits [31:28] of PCI PIO AFSR should be looked at by software only if one of bits[63:59] are set . In case none of bits[63:59] are set, bits [31:28] should be ignored.

TABLE 4-39 PCI PIO AFAR(PCI_CSRBase+0x0.0000.2018)

Field	Bits	Reset	Description	R/W
Reserved	63:32	0	Reserved, read as 0.	R
PA_Offset	31:0	0	Offset of physical address of transfer in Error	R

4.5.1.3 PCI Diagnostic Register

TABLE 4-40 PCI Diagnostic Register(PCI_CSRBase+0x0.0000.2020)

Field	Bits	Reset	Description	R/W
Reserved	63:11	0	Reserved, read as 0.	R
Reserved	10	0	Reserved, read as 0.	R
DIS_BYPASS	9	0	Disable MMU Bypass mode When set to 1, JIO will not respond to PCI Dual-Address Cycles, thus disallowing MMU bypass mode.	R/W
DIS_TTO	8	0	Disable TRDY# timeout errors. When set to 1, JIO will not terminate any transactions as master due to excessive delays in TRDY#.	R/W
Reserved	7	0	Reserved, read as 0.	R
DIS_RETRY	6	0	Disable retry limit. When set to 1, JIO will not abort PIO operations after 16,384 retries, but will continue indefinitely.	R/W
DIS_INTSYNC	5	0	Disable DMA write / interrupt synchronization. When set to 1, interrupts will not wait until associated DMA is complete before proceeding.	R/W
Reserved	4	0	Reserved, read as 0.	R
I_DMA_D_PAR	3	0	Invert DMA data parity 0 = Correct parity asserted 1 = Incorrect parity asserted for all PCI DMA read data phases. Both the regular parity signal and the 64-bit parity extension are affected.	R/W
I_PIO_D_PAR	2	0	Invert PIO data parity 0 = Correct parity asserted 1 = Incorrect parity asserted for all PCI PIO write data phases.	R/W
I_PIO_A_PAR	1	0	Invert PIO address parity 0 = Correct parity asserted 1 = Incorrect parity asserted for all PCI PIO address phases.	R/W
Reserved	0	0	Reserved, read as 0.	R

4.5.1.4 PCI Target Retry Limit Register

Each time a PCI target unit has to signal Retry on any DMA read or write, it increments a counter. This counter is cleared by hardware at any time that the target unit transfers data. It is also cleared if the retry limit has been crossed and the Target unit signals Target Abort.If the counter equals the Target Retry Limit for this PCI bus, and the Target Retry Limit is not 0, then the DMA read transaction is not Retried, but a Target Abort is signalled. The Target Retry Limit can apply to both DMA reads and writes, although only the reads should need its effect.

There may be PCI master latency timers that could kick in, but we don't depend on them, since implementation of them in PCI devices is uncertain.

TABLE 4-41 PCI Target Retry Limit Register (PCI_CSRBase+0x0.0000.2030)

Field	Bits	Reset	Description	R/W
Rsvd	63:24	0x00	Reserved. Read as zeroes	R
PCI_TRGT_RTRY_LMT_HI	23:4	0x00	Programmable portion of retry limit register.	R/W
PCI_TRGT_RTRY_LMT_LO	3:0	0x00	Read only portion of retry limit register. Hardwired to 0.	R

This register specifies the value of the PCI Target Retry Limit Register. Only bits [23:4] are programmable, giving a timer granularity of 16 retries. The bottom four bits will read as 0.

Note – Programming bits [23:4] to 0x00 would cause unlimited retries, and if the target never returns data or the writes never drain, the system would hang.

4.5.1.5 PCI Target Read Latency Timer

This timer is used to limit the number of wait states that the PCI target can insert to the first data phase for a read. A counter is incremented for each wait state. If the counter reaches the Timer value and the value is non-zero and the transaction has not begun on the destination bus, then the transaction is Retried. This then allows the bus originating bus to be used by other masters.

Retries caused in this manner are added to the target retry counter, but they are not prevented by the retry counter hitting the Target Retry Limit. (Note: The target retry counter saturates at the Target Retry Limit; it will not count beyond it, and JIO would assert Target Abort Error.)

TABLE 4-42 PCI Target Read Latency Timer Register (PCI_CSRBase+0x0.0000.2038)

Field	Bits	Reset	Description	R/W
Rsvd	63:5	0x00	Reserved. Read as zeroes	R
PCI_TRGT_LTNCY_LMT	4:0	0x0B	Programmable portion of latency timer.	R/W

This register specifies the value of the PCI Target Read latency timer.

4.5.1.6 Interrupt Routing Register

This register is located in the PCI A Leaf only. It would be initialized by OBP to route the interrupts into PCI-A bus or PCI-B bus.This register is invisible to Unix OS. The register contains 64 bits, one bit per INO. If set to 0, indicates that the Interrupt for the corresponding INO is routed to PCI-A bus (in first cycle of Interrupt Packet on JBus,JPID of Source Port/IGN = {JIO's JPID[4:1],1'b0} *for PCI*) , if set to 1, routed to PCI-B bus (in first cycle of Interrupt Packet on JBus,JPID of Source Port/ IGN = {JIO's JPID[4:1], 1'b1} *for GR*).

TABLE 4-43 Interrupt Routing Register (PCI-A_CSRBase + 0x0.0000.2240)

Bits	Reset	Description	R/W
63:0	0x00	One bit per INO. If 0, corresponding Interrupt to be mapped to PCI bus A. If 1, corresponding Interrupt to be mapped to PCI bus B.	R/W

4.5.1.7 PCI Target Address Space Register

The PCI Target Address Space Register selectively enables 512 MByte regions as target PCI addresses for JIO. JIO examines single-cycle PCI addresses and responds

Table 4-44 PCI Target Address Space Register (PCI_CSRBase+0x0.0000.2490 & PCI_ConfigBase+0x54)

Field	Bits	Description	Reset	RW
Reserved	63:8	Reserved, read as 0.	0	R0
EF_enable	7	Respond to 0xE000.0000-0xFFFF.FFFF	0	R/W
CD_enable	6	Respond to 0xC000.0000-0xDFFF.FFFF	0	R/W
AB_enable	5	Respond to 0xA000.0000-0xBFFF.FFFF	0	R/W
89_enable	4	Respond to 0x8000.0000-0x9FFF.FFFF	0	R/W
67_enable	3	Respond to 0x6000.0000-0x7FFF.FFFF	0	R/W
45_enable	2	Respond to 0x4000.0000-0x5FFF.FFFF	0	R/W
23_enable	1	Respond to 0x2000.0000-0x3FFF.FFFF	0	R/W
01_enable	0	Respond to 0x0000.0000-0x1FFF.FFFF	0	R/W

as a target if address[31:28] select an enabled region. Dual-cycle addresses are not selectively enabled as a target for JIO. Only address[63:50]==0x3fff indicates JIO target.

Note that more than one region can be enabled, and holes are allowed. No other PCI device should be enabled to respond to the JIO target address space.

Note that this register is accessible both from the CPU side (address:PCI_CSRBase+0x0.0000.2490, 8 byte access) and the PCI side (config address 54, 1 byte access).

At reset accesses to all regions are disabled. Then either a PCI device or a Sparc IIIi CPU writes a 1 to one or more bits enabling access to that range from the PCI side.

Implementation Note:

Note – The target space logic should be implemented using pci_ad[31:29] as the mux selects on an 8:1 mux, with the 8 bit value (PCI Target Address Space Register bits [7:0]) feeding the 8 mux inputs, and the mux output indicating “target hit”.This allows programmability with just one level of logic for hit detect.

4.5.1.8 PCI Target Error VA Log Register

This register logs the Virtual Address of the PCI access to JIO corresponding to an Address Parity Error or Data Parity Error detected by JIO or Target Abort issued by JIO, JIO being the target for the transaction.This register is provided only for the aid of software debug.No address logging happens however for PERR or SERR detected by other PCI devices / agents.

The conditions under which JIO target aborts and sets STA bit in PCI Config Status Register(PCI_ConfigBase + 0x06) and logs the Virtual Address in the PCI Target Error VA Log Register are :

- JIO detects address parity error
- JIO terminates a transaction with Target Abort if the FRC_TRGT_ABRT (bit 27 of the PCI control and status) is set to 1
- JIO terminates a DMA Read transaction with Target Abort if the retry number cross the Retry Limit number specified by the PCI Target Retry Limit Register

---

Note – This register gets updated when a parity error gets detected by JIO as a target or when Target Abort gets issued by JIO as a target. If there are errors of different types (e.g : address parity error and data parity error both occured) logged, this always has the address of the first error, and software would not know which error happened first, and hence which error the address logged in this register corresponds to. For a single error logged, it would not be possible to tell if one error or multiple errors of the same type occurred, but the address is guaranteed to be of the first error.

Note – Due to a bug in JIO, it does not set the STA bit and log the virtual address into PCI Target Error VA Log register when it terminates a DMA read transaction with Target Abort if the return data from JBUS contains UE . Although this behavior is in violation of PCI spec , this bug will not be fixed in JIO as the JBUS Cluster is able to log the UE information in UE AFSR / AFAR registers and dispatch the interrupt to inform SW if it detects UE in the DVMA transactions.

Note – Note that the reset value of this register is 0x0. This should be fine provided software looks at this register only after the occurence of the error, and the system does not get reset between the time the error occurs and software examines the contents of this register.

TABLE 4-45 PCI Target Error VA Log Register(PCI_CSRBase+0x0.0000.2498)

Field	Bits	Reset	Description	R/W
Error VA	63:0	0x0	Virtual Address for the PCI transaction causing Error with JIO as the target. For DAC : 63:0 = PCI Address[63:0] For SAC : 31:0 = PCI Address[31:0], 63:32 = Don't Care	R

4.5.1.9 PBM Configuration Space

The PBM also contains a configuration header whose format is specified by the PCI Specification. The registers in the configuration header are accessed in PCI Configuration Address Space. The PBM is considered to be device 0, function 0 on its PCI bus. After a reset, the PBMs Bus Number register is set to 0

TABLE 4-46 Default Addresses PCI Leaf's Internal PCI Configuration Registers

Register	Address
PBM Config Registers	PCI_ConfigBase+0x000000 - PCI_ConfigBase+0x0000FF

Note – These are the addresses in effect after reset. However, since the PCI bus number of the PBM can be changed by software, the actual offset for these spaces may be different than what is listed above.

Note – The PCI Configuration Address Space is a little-endian address space. When accessing configuration space registers, software should take advantage of one of the SPARC V9 little-endian support mechanisms to get proper byte ordering. These mechanisms include little-endian ASIs or MMU support for marking pages little-endian.

The table below lists the configuration header registers, as defined by the PCI specification and PCI System Design Guide. Several of the registers are not implemented in JIO which is indicated by shading in the table. The rule used is that any optional register for which equivalent information exists elsewhere is not implemented.

TABLE 4-47 Configuration Space Header Summary

Register	Offset	Size
Required PCI device configuration header:		
Vendor ID	0x00	2 bytes
Device ID	0x02	2 bytes
Command	0x04	2 bytes
Status	0x06	2 bytes
Revision ID	0x08	1 byte
Programming I/F Code	0x09	1 byte
Sub-class Code	0x0A	1 byte
Base Class Code	0x0B	1 byte
Cache Line Size	0x0C	1 byte
Latency Timer	0x0D	1 byte
Header Type	0x0E	1 byte
BIST	0x0F	1 byte
Base Address	0x10-0x27	Varies
Reserved	0x28-0x2F	n/a
Expansion ROM	0x30	4 bytes
Reserved	0x34-0x3B	n/a
Interrupt Line	0x3C	1 byte
Interrupt Pin	0x3D	1 byte
MIN_GNT	0x3E	1 byte
MAX_LAT	0x3F	1 byte
Optional bridge configuration header:		
Bus Number	0x40	1 byte
Subordinate Bus Number	0x41	1 byte
Reserved	0x42-0xFF	n/a
Disconnect Counter	Unspecified	1 byte

TABLE 4-47 Configuration Space Header Summary

Register	Offset	Size
Bridge Command/Status	Unspecified	4 bytes
Bridge Memory Base Address	Unspecified	4 bytes
Bridge Memory Limit Address	Unspecified	4 bytes
DOS Read Attributes	Unspecified	2 bytes
DOS Write Attributes	Unspecified	2 bytes
Bridge I/O Base Address	Unspecified	2 bytes
Bridge I/O Limit Address	Unspecified	2 bytes

Note – The sizes listed in the table above are just the logical size for each register. Actual PIO access to the registers can be in any size from 1 to 4 bytes, provided the access doesn't span multiple 32-bit words.

Extension Register:

In order to enable one JIO to configure another JIO without the CPUs present (test mode bringup), PCI_A leaf needs to support a PCI config write to one register called Extension Register from the PCI side. This register contains a 11 bit "Extension" value that forms the upper address bits for the 43 bit Physical Address to Jbus while the "Extension Enable" bit is set, SAC with IOMMU disabled.Note that this register is only visible from the PCI side of the chip, and it is invisible from the Sparc IIIi side.

Register	Address	Size
Extension Register	0x50	2 bytes

TABLE 4-48 Extension Register (PCI-A_ConfigBase+ 0x50)

Field	Bits	Reset	Description	R/W
Reserved	15:12	0x00	Reserved. Read as zero's	R
Extension Enable	11	0	When set enables extension.	R/W
Extension	10:0	0x00	Extension value that would form PA[42:32] in Extension Mode.	R/W

Vendor ID(PCI_ConfigBase + 0x00)

Read only, VendorID<15:0> = 0x108E.

Device ID(PCI_ConfigBase + 0x02)

Read only, DeviceID<15:0> = 0xA801.

Command Register

TABLE 4-49 Command Register(PCI_ConfigBase + 0x04)

Field	Bits	Reset	Description	R/W
Reserved	15:10	0	Reserved, read as 0.	R
FAST_EN	9	0	Enable fast back-to-back cycles to different targets. Hardwired to 0 (disabled).	R
SERR_EN*	8	0	Enable driving of SERR# pin when 1.	R/W
WAIT	7	0	Enable use of address/data stepping Hardwired to 0 (disabled).	R
PER*	6	0	Enable reporting of parity errors only when 1.	R/W
VGA	5	0	Enable VGA palette snooping Hardwired to 0 (disabled).	R
MWI	4	0	Enables use of Memory Write & Invalidate Hardwired to 0 (disabled).	R
SPCL	3	0	Enables monitoring of special cycles Hardwired to 0 (disabled).	R
MSTR	2	1	Enables ability to be bus master Hardwired to 1 (enabled).	R
MEM	1	1	Enables response to PCI MEM cycles Hardwired to 1 (enabled).	R
IO	0	0	Enables response to PCI I/O cycles. Hardwired to 0 (disabled).	R

*PER and SERR_EN bits control Data Parity and Address Parity Checking as follows :

PER	SERR_EN	Parity Check
0	0	PCI address and data parity checking fully disabled in JIO
0	1	PCI address and data parity checking fully disabled in JIO
1	0	PCI data parity checking enabled in JIO (including side effects such as interrupt generation,PERR# assertion,BERR on PIO Read to CPU)
1	1	PCI address and data parity checking enabled in JIO (including side effects such as interrupt generation,target abort assertion, SERR#/PERR# assertion,BERR on PIO Read to CPU)

Status Register

TABLE 4-50 Status Register(PCI_ConfigBase + 0x06)

Field	Bits	Reset	Description	R/W
DPE	15	X	Set if PBM detects a parity error	R/W1C
SSE	14	0	Set if PBM signalled a system error. This occurs if the PBM detects a PCI address parity error .	R/W1C
RMA	13	0	Set if PBM generates a master-abort	R/W1C
RTA	12	0	Set if PBM receives a target-abort	R/W1C
STA	11	0	Set if PBM generates target-abort	R/W1C
DVSL	10:9	0x1	Timing of DEVSEL#. Hardwired to 01 (medium speed response)	R
DPAR	8	0	Set when parity error occurs while PBM is bus master, if PER in command register also set.	R/W1C
FASTCAP	7	1	Indicates ability to accept fast back-to-back cycles as target, when the back-to-back transactions are not to the same target. Hardwired to 1 (allowed)	R
UDF_SUPPORT	6	0	User Definable Feature Support Hardwired to 0 (no user definable features)	R
66MHZ_CAPABLE	5	-	Indicates ability to run at 66MHz clock speed. Hardwired to 1 (66MHz capable) for PBMA and 1 for PBMB.	R
Reserved	4:0	0	Reserved, read as 0	R

Revision ID Register (PCI_ConfigBase + 0x08)

Read only, RevisionID<7:0> = 0x00. This register will always read as 0. The actual revision number for JIO is contained in the JIO Control/Status Register.

Programming I/F Code Register (PCI_ConfigBase + 0x09)

Read only, ProgrammingIFCode<7:0> = 0x00.

Sub-class Code Register(PCI_ConfigBase + 0x0A)

Read only, SubclassCode<7:0> = 0x00. (Specifies host bridge device).

Base Class Code Register(PCI_ConfigBase + 0x0B)

Read only, BaseClassCode<7:0> = 0x06. (Specifies bridge device).

Latency Timer Register

This 8-bit read / write register specifies the value of the latency timer for the PBM as a bus master. Only the top five bits are implemented, giving a timer granularity of 8 PCI clocks. The bottom three bits will read as 0 and should be written as 0.

TABLE 4-51 Latency Timer Register(PCI_ConfigBase + 0x0D)

Field	Bits	Reset	Description	R/W
LAT_TMR_HI	7:3	0x04	Programmable portion of latency timer.	R / W
LAT_TMR_LO	2:0	0	Read only portion of latency timer. Hardwired to 0.	R

Header Type Register

TABLE 4-52 Header Type Register(PCI_ConfigBase + 0x0E)

Field	Bits	Reset	Description	R/W
MULTI_FUNC	7	0	Indicates whether the PBM is a multi-function PCI device. Hardwired to 0 (not multi-function).	R
HDR_TYPE	6:0	0	Defines layout of configuration header bytes 0x10-0x3F. Hardwired to 0 (the only defined value in PCI specification)	R

Bus Number(PCI_ConfigBase + 0x40)

This 8-bit read / write register specifies the number of the PCI bus this bridge resides on. Its value upon reset is 0.

Subordinate Bus Number(PCI_ConfigBase + 0x41)

This 8-bit read / write register specifies the highest subordinate bus number beneath this bridge. Its value upon reset is 0.

Unimplemented Registers

The following registers are defined in the PCI Specification or PCI System Design Guide, but are not implemented in JIO's PBMs for the indicated reasons.

Cache Line Size - The cache line size is fixed at 64-bytes by the Jbus architecture.

BIST - Built-In-Self-Test is not implemented in JIO in a way that would need to be controlled via PCI Configuration registers.

Base Address Registers - The bridge itself has neither memory nor I/O space. Its configuration space is accessible only from the host and is hard-mapped.

Interrupt Line, Interrupt Pin - Do not apply. External PCI interrupt lines are handled by the RISC asic, while internal PBM interrupts are signalled directly to the Mondo block, and are not signalled on PCI interrupt lines.

Min_Gnt, Max_Lat - There is no regular traffic pattern to programmed I/O. Values of zero indicate there are no stringent requirements (true).

Disconnect Counter - This seems to be intended mainly for cases where the other bus (host bus in this case) is potentially very slow. This shouldn't apply to Jbus.

Bridge Memory/IO Base and Limit Address - These registers are defined for an entirely flat address space which the Jbus and JIO cannot abide by.

DOS Attribute Registers - DOS compatibility is not a feature of JIO.

4.5.2 I/O Cache Registers

I/O Cache Register Summary

JIO contains two nearly identical copies of an I/O cache. The programmer's model for both copies is identical. Register addresses are shown with respect to the base address PCI_CSRBase, whose real value should be one of the two base addresses PCI-A_CSRBase or PCI-B_CSRBase.

TABLE 4-53 Offset of I/O Cache Registers

Register	Address	Access Size
I/O Cache Control/Status Register(NEW)	PCI_CSRBase+0x0.0000.2248	8 bytes
I/O Cache Tag Diagnostic Registers (NEW)	PCI_CSRBase+0x0.0000.2250 - 0x0000.2288	8 bytes
I/O Cache Data Ram Diagnostic Registers (NEW)	PCI_CSRBase+0x0.0000.2290 - 0x0000.2488	8 bytes

4.5.2.1 I/O Cache Control/Status Register

TABLE 4-54 I/O Cache Control/Status Register (PCI_CSRBase+0x0.0000.2248)

Field	Bits	Reset	Description	R/W
Rsvd	63:20	0x00	Reserved. Read as zeroes.	R
WRT_PEN	19	0	Enable prefetch for Partial Line Writes when set to 1.	R/W
NC_PEN_RD_MLTPL	18	0	Enable prefetch in I/O cache if PCI command is “Memory Read Multiple” to Non-Cacheable address space. 0: Disable 1: Enable	R/W
NC_PEN_RD_ONE	17	0	Enable prefetch in I/O cache if PCI command is “Memory Read” to Non-Cacheable address space. 0: Disable 1: Enable	R/W
NC_PEN_RD_LINE	16	0	Enable prefetch in I/O cache if PCI command is “Memory Read Line” to Non-Cacheable address space. 0: Disable 1: Enable	R/W
PLEN_RD_MLTPL	15:14	0x00	Number of lines to prefetch for RD_MULTPL starting from POFFSET. 0x0 corresponds to 1 0x3 corresponds to 4. only 0x0 to 0x2 valid.	R/W
PLEN_RD_ONE	13:12	0x00	Number of lines to prefetch for RD_ONE starting from POFFSET. 0x0 corresponds to 1 0x3 corresponds to 4. only 0x0 to 0x2 valid.	R/W
PLEN_RD_LINE	11:10	0x00	Number of lines to prefetch for RD_LINE starting from POFFSET. 0x0 corresponds to 1 0x3 corresponds to 4. only 0x0 to 0x2 valid.	R/W
POFFSET	9:3	0x00	Prefetch offset as a multiple of 64 bytes. Should be within 8K page. 0x00 corresponds to 1 0x7e corresponds to 127 0x7f is illegal. JIO would treat it as 0x00 (prefetch offset of 1).	R/W

TABLE 4-54 I/O Cache Control/Status Register (PCI_CSRBase+0x0.0000.2248)

Field	Bits	Reset	Description	R/W
C_PEN_RD_MLTPL	2	0	Enable prefetch in I/O cache if PCI command is “Memory Read Multiple” to Cacheable address space. 0: Disable 1: Enable	R/W
C_PEN_RD_ONE	1	0	Enable prefetch in I/O cache if PCI command is “Memory Read” to Cacheable address space. 0: Disable 1: Enable	R/W
C_PEN_RD_LINE	0	0	Enable prefetch in I/O cache if PCI command is “Memory Read Line” to Cacheable address space. 0: Disable 1: Enable	R/W

Note – When WRT_PEN field is set to 1, depending on the last Read command dispatched to JIO(Read,Read line , or Read Multiple) ,JIO uses the value programmed in the corresponding PLEN field as the number of lines to prefetch for partial line writes. If there is no prior read, JIO uses the value programmed in PLEN_RD_ONE field as the number of lines to prefetch for partial line writes.

Note – PA[42] from the IOMMU is looked at to decide whether to prefetch based on {C_PEN_RD_MLTPL,C_PEN_RD_ONE,C_PEN_RD_LINE} or {NC_PEN_RD_MLTPL,NC_PEN_RD_ONE,NC_PEN_RD_LINE}.

4.5.2.2 I/O Cache Tag Diagnostic Register

The I/O cache tags are made accessible for diagnostic purposes.There is one register of each of the 8 entries in the I/O cache.

TABLE 4-55 I/O Cache Tag Diagnostic Register(PCI_CSRBase+ {0x0.0000.2250 - 0x0000.2288})

Bits	Fields	Description	Reset Value	Type
63:60	Rsvd	Reserved. Read as zero	0	R
59	Rsvd	Reserved. Read as zero	0	R
58:56	S	Cache line state. Valid encodings are: I = 3'b000 S = 3'b001 O = 3'b101 M = 3'b111	0xX	R/W
55:42	Rsvd	Reserved. Read as zero	0x0	R
41:6	PA[41:6]	Physical address	0xXX XXXX XXXX	R/W
5:0	Rsvd	Reserved. Read as zero.	0x0	R

S identifies the cache line state of the entry. If software needs to mark a line as dirty, it can make its state O or M.

Note – There are 8 separate tag diagnostic registers (one for each entry of the I/O cache) aligned on 8 byte boundaries.

Note – It is illegal for Software to program the S field in I/O Cache Tag Diagnostic Registers as M (3'b111) while Disable_M_State bit (bit 53 of JIO Control and Status Register(Jbus_Base + 0x41.0000)) has been programmed to 1'b1. Such a programming will not be blocked by JIO but will lead to data corruption issues in the system.

4.5.2.3 I/O Cache Data RAM Diagnostic Register

There are 8 cache lines in the I/O cache. All of this data is accessible via the I/O cache Data RAM Diagnostic Registers.There are 8 registers for each of the 8 entries in the I/O cache.

TABLE 4-56 I/O Cache Data RAM Diagnostic Register (PCI_CSRBase+ {0x0.0000.2290 - 0x0000.2488})

Bits	Fields	Description	Reset Value	Type
63 : 0	DRDA	Data	X	R/W

4.5.2.4 I/O Cache Flush to Memory

There is no way for OS to flush I/O cache in normal operation unless we know what addresses might be in the I/O cache. In case we do not know the address, only way to do that would be is to first evict all dirty lines in the I/O cache by doing a bunch of DMA reads (displacement flush). Then diagnostic PIO stores can be used to invalidate clean lines. While the PIO stores are occurring, any new DMA may cause the clean lines to go dirty again. Hence the stores have to be issued only when no I/O DMA activity is in progress.

Alternatively if the address is known, CPU can execute a CAS that does not swap. It will get dirty copy from the I/O cache into the CPU cache. Note that here the flush does not go to memory directly, but the CPU can later evict it to memory.

Note – The I/O cache is always as coherent as any of the CPU caches. So the expectation will be that Solaris would not need to flush the I/O cache during normal operation.

4.5.3 IOMMU Registers

4.5.3.1 Translation Storage Buffer Overview

The IOMMU fetches translation information from a Translation Storage Buffer (TSB) in memory. The TSB contains one-level mapping information for the virtual DMA pages. A single TSB entry is called Translation Table Entry (TTE), and takes 8 bytes.

JIO supports several TSB table sizes, specified by the TSB_SIZE field of IOMMU Control Register. TSB table sizes supported are 1K, 2K, 4K, 8K, 16K, 32K, 64K and 128K entries (not bytes), which allows a DVMA address space of 8M to 1G using 8K pages, and 64M to 2G using 64K pages (a DVMA address space larger than 2G is not supported, so 128K and 64K TSB sizes are not supported with a 64K page size). Software must set up TSB before it allows translation to start.

Note – Only CPUs are allowed to initialize IOMMU TSB in memory. PCI devices are not allowed to initialize IOMMU TSB in memory, but can read contents of TSB in memory.

Translation Table Entry

Translation Table Entries (TTE) contain translation information for virtual pages. The IOMMU hardware reads one TTE during a table walk and stores it in the Translation-Lookaside Buffer (TLB). A TTE entry has valid information only when bit DATA_V is set. Information stored in the TTE has the following format:

TABLE 4-57 TTE Data Format

Field	Bits	Description
DATA_V	63	Valid bit (1 = TTE entry has valid mapping)
Reserved	62	Reserved.
DATA_SIZE	61	Page size of the mapping (0 = 8K, 1 = 64K)
Reserved	60	Reserved.
LOCALBUS	59	Local Bus bit. Not used
CONTEXT	58:47	Context number for this page, used in flushing multiple related pages.
Reserved	46:43	Reserved.
DATA_PA	42:13	Contains bits <42:13> of physical address. Bits 15:13 are not used for 64K page.
DATA_SOFT	12:7	Reserved for software use.
Reserved	6:5	Reserved.
CACHEABLE	4	Cacheable (1 = cacheable page, 0 = non-cacheable page)
Reserved	3:2	Reserved.
DATA_W	1	Set if this page is writeable
Reserved	0	Reserved.

Note – The LOCALBUS bit is not meaningful in a PCI environment, and is not stored in the TLB. The MMU hardware drops this bit after a table walk.

By looking at the CACHEABLE bit and the Read Command from the PCI Bus, the I/O caches in JIO will initiate prefetch cycles on Jbus.

Table Walk

During an IOMMU table walk (TSB lookup) the physical address for the TTE entry that will be fetched is calculated from the DVMA address (VirtAddr) as follows:

```
Table Size = 2 ^ (TSB_SIZE + 10)
Page Bits = (TBW_SIZE == 1) ? 16 : 13
Entry# = (VirtAddr >> Page Bits) & (Table Size - 1)
TTE Address = TSB_BASE + 8 * Entry#
```

The TSB is always accessed using cacheable Jbus transactions.

IOMMU Register Summary

TABLE 4-58 Addresses of IOMMU Registers

Register	Address	Access Size
IOMMU Control Register	PCI_CSRBase+0x00.0200	8 bytes
TSB Base Address Reg	PCI_CSRBase+0x00.0208	8 bytes
IOMMU Flush Page Register	PCI_CSRBase+0x00.0210	8 bytes
IOMMU Flush Context Register	PCI_CSRBase+0x00.0218	8 bytes
Translation Fault Address Register	PCI_CSRBase+0x00.0220	8 bytes
TLB Compare Setup Diag Reg	PCI_CSRBase+0x00.A400	8 bytes
TLB Compare Result Diag Reg	PCI_CSRBase+0x00.A408	8 bytes
IOMMU LRU Queue Diag Regs	PCI_CSRBase+0x00.A500 - PCI_CSRBase+0x00.A57F	8 bytes
TLB Tag Diag Regs	PCI_CSRBase+0x00.A580 - PCI_CSRBase+0x00.A5FF	8 bytes
TLB Data RAM Diag Regs	PCI_CSRBase+0x00.A600 - PCI_CSRBase+0x00.A67F	8 bytes

4.5.3.2 IOMMU Control Register

The Control Register provides means to enable and disable the IOMMU and diagnostic mode, and set the TSB size and page size.

TABLE 4-59 IOMMU Control Register(PCI_CSRBase+0x00.0200)

Field	Bits	Reset	Description	Type
RESERVED	63:29	0	Reserved, read as zeros	R
ERR_BAD_VA	28	0	For TSB and TBW defined VA spaces (please refer to Table 4-60) smaller than 512 Mbytes, this error is detected if the PCI DMA virtual address is outside the range created by aligning the TSB and TBW defined VA space to the top of the 512 Mbyte chunks of enabled Target Address Space	R/W1C
ERR_ILLEGAL_TSB_TBW	27	0	IOMMU Error due to Illegal combination of TSB_SIZE and TBW_SIZE values.	R/W1C

TABLE 4-59 IOMMU Control Register(PCI_CSRBase+0x00.0200)

Field	Bits	Reset	Description	Type
ERRSTS	26:25		If ERR is set, indicates the type of error logged in the IOMMU state Error Status: 00 = Protection Error 01 = Invalid Error 10 = Timeout 11 = ECC Error (UE)	R/W1C
ERR	24		Set when IOMMU is written with an Error. This bit gets set when any of following errors happen : - Out of Range - Illegal TSB_TBW - Protection - Invalid - Timeout - ECC Error (UE)	R/W1C
LRU_LCKEN	23	0	LRU Lock Enable Bit. When set, only the TLB entry specified by the Lock Pointer can be replaced.	R/W
LRU_LCKPTR	22:19	X	LRU Lock Pointer. Works in conjunction with the LRU Lock Enable bit to limit TLB replacement to a single entry.	R/W
TSB_SIZE	18:16	X	TSB table size measured in the number of 8 byte entries. 0=1K, 1=2K, 2=4K, 3=8K, 4=16K, 5=32K, 6=64K, 7=128K.	R/W
RESERVED	15:6	0	Reserved, read as zeros	R

TABLE 4-59 IOMMU Control Register(PCI_CSRBase+0x00.0200)

Field	Bits	Reset	Description	Type
SEG_DISP[1:0]	5:4	0x0	This field is valid only for 1GB and 2GB sized TSB and TBW defined VA spaces. For < 1 GB TSB and TBW defined VA space, the value in this field is ignored by IOMMU. For TSB and TBW defined VA spaces of 1GB and 2GB, these bits allow the enabled PCI Target Address Space to start on any 512 mbyte boundary, The TSB Base Address is still required to be on a natural boundary of 1GB or 2GB. By creating a 2-bit adder with these bits and VA[30:29] before creating the TSB index, the linear Target Address Space is mapped correctly to the linear TSB range. These bits should be programmed to capture the difference between the TSB base address [30:29], and the VA[30:29] implied by the beginning enabled Target Address Space (which is on 512 MB granularity). See Fig 4-6 for programming directions.	R/W
RESERVED	3	0	Reserved, read as zero	R
TBW_SIZE[1]	2	X	Assumed page size during TSB lookup. 0 = 8K page 1 = 64K page	R/W
MMU_DE	1	0	Diagnostic mode enable, when set it enables the diagnostic mode. See description of TLB tag diagnostics.	R/W
MMU_EN	0	0	IOMMU enable bit, when set it enables translations.	R/W

1. If DVMA mappings are always 8K pages, or mixed 8K and 64K pages, set this bit to '0' so that the index is constructed for 8K lookup. If all DVMA mappings are to 64K pages, set this bit to '1' so that the index is based on 64K pages. When this bit is '0', a 64K mapping should be placed in all 8 TSB entries in which it is indexed.

Invalid TTE, DMA UE , TTE Read Timeout Errors

For an IOMMU miss, if the returned TTE data has Valid = 0 or has a timeout or has an uncorrectable ECC Error (UE), the IOMMU installs the bad VA into the CAM and and modifies ERRSTS field and sets ERR =1 and Valid =1. The IOMMU also sets the ERR bit and adjusts the ERRSTS[1:0] bits in the IOMMU Control and status register to reflect the occurence of the error.

The valid bit for the entry is set, regardless of the state of the valid bit in the TTE data, so that the DMA transaction does not cause another IOMMU miss.

Any future read access to the IOMMU CAM that hits in that entry with ERR and the ERRSTS bits still set and Valid bit = 1, would cause errors and Target aborts to be issued to the PCI device. For a future write access to the IOMMU CAM that hits in that entry with ERR and the ERRSTS bits still set and Valid bit = 1 , just causes the error to be logged in the IOMMU Control and Status Register , but does not cause any target aborts.

Software is responsible for flushing the IOM entry when it rectifies the missing TSB entry or bad DMA address.

Protection Error

For an IOMMU miss on a DMA write , if the returned TTE data lacks the appropriate write privilege,the IOMMU installs the VA into the CAM with ERR = 0 and Valid = 1. However IOMMU sets the ERR bit and adjusts the ERRSTS[1:0] bits in the IOMMU Control and status register to reflect the occurence of the protection error.

A future read that hits in the same entry gets serviced by the IOMMU as a normal hit.

A future write that hits on the same entry will be treated as an error and cause an interrupt. The ERR bit will be set and ERRSTS[1:0] bits modifeid to 0x00 in IOMMU Control and Status Register to indicate Protection Error, if and only software has cleared the ERR bit due to any prior errors.

Note – The Error Overwrite Policy for IOMMU Errors is that in case of multiple errors happening , only the first error gets logged in the IOMMU Control and Status Register. The second error cannot get logged and the ERRSTS, ERR_BAD_VA, ERR_ILLEGAL_TSB_TBW bits cannot get appropriately set , unless software has cleared the ERR bit in this register for the primary error . An interrupt gets raised however (for all IOMMU errors other than UE and Timeout) on occurrence of the primary error. For UE and Timeout , Jbus cluster of JIO asserts error interrupt.

Target Abort and Interrupt

For all IOMMU errors detected other than Timeout and ECC(UE) Errors, an interrupt is raised to the Interrupt Controller. Then software comes and examines the cause of the Error, takes appropriate action and clears the ERR bit.For Timeout and ECC(UE) Error on a TTE fetch, Jbus interface in JIO logs the error and asserts interrupts.

For a DMA Read, the IOMMU Error is reported to the DMA master as a Target Abort; the PBM logs its target-abort generation with the STA bit in the PCI Configuration Space Status Register.This should also be cleared by the error software.

TSB sizing and Target Address Space configurations

SUPCISUPCI assumed that the DMA addresses start at 0xFFFF.FFFF and work down. The legal ranges were based on that and the combination of TSB_SIZE and TBW_SIZE.

However, JIO, like Sabre, introduces a programmable DMA target address space on the PCI bus.

Normally PCI devices are assigned target addresses spaces at very fine granularity, using Base Address Registers (BARs) in the PC nomenclature. A starting address is assigned for each target. (Normally the supported granularity of the BAR matches the amount of space the device needs...so you align on boundaries equal to the space required).

The target has a register which tells software how much space it needs starting at the BAR, so software doesn't program another BAR to conflict with the space that device will respond to.

PCI cards can have multiple BARs, so they can support non-contiguous target address spaces.

JIO and Sabre, for decode logic simplicity, restrict the target space to 8 individual enables for 512 Mbyte chunks. This is like having 8 BARS with fixed addresses, which can be individually enabled.

Programmable Target Address Space implications

Following Sabre's lead, JIO has the PCI Target Address Space Register that allocates eight 512 Mbyte chunks of address space (4 Gbyte total in the 32 bit single address cycle address space. Dual address cycles have a fixed target mapping for JIO, i.e. address[63:50] == 0x3fff.)

The eight segment enable bits in this register means that the 512 Mbyte chunks can be selectively assigned in a non-contiguous manner only for single address cycles. In Solaris, all DMA is set up to use the IOMMU. (it can be bypassed, though).

This means that the TSB organization, which has a contiguous index range as shown in Table 4-60, wants the 512 Mbyte chunks to be assigned contiguously, and in most cases, on the natural boundary of the total target space assigned.An enhancement for TSB and TBW defined VA spaces of 1GB and 2GB allows starting on the Target Address Space on any 512mbyte boundary. See Figure 4-6 for details. If the TSB and TBW defined VA space is smaller than the Target address space assigned, then the TSB base address can be on any TSB size natural boundary.

TABLE 4-60 TSB Indexing

	TBW_SIZ == 0		TBW_SIZ == 1	
TSB_SIZE	VA Space Size	TSB Index [3]	VA Space Size	TSB_Index [3]
0	8 MB	VA<22:13>	64 MB	VA<25:16>
1	16 MB	VA<23:13>	128 MB	VA<26:16>
2	32 MB	VA<24:13>	256 MB	VA<27:16>
3	64 MB	VA<25:13>	512 MB	VA<28:16>
4	128 MB	VA<26:13>	1 GB	VA<29:16>
5	256 MB	VA<27:13>	2 GB	VA<30:16>
6	512 MB	VA<28:13>	not allowed[1]	--
7	1GB	VA<29:13>	not allowed[1]	--

1. Hardware does not prevent illegal combinations from being programmed. If an illegal combination is programmed into the IOMMU, all translation requests will be rejected as invalid.

So there should be no need to have noncontigous bit patterns in the target address space register.

Out of Range Virtual Address Error

The PCI spec allows devices to reserve more target address space then they actually respond to. This allows simplification of target address space decode logic.

However, JIO can differentiate between supported addresses (defined by the TSB and TBW size) and addresses within the claimed region but not supported by the TSB and TBW.

The combination of TSB_SIZE and TBW_SIZE determines the range of valid virtual addresses as shown in Fig 4-4 and Fig 4-5.Most of the TSB/TBW sizes imply use of target address spaces that are < 512 Mbytes. It is assumed that the use of the region is from top to bottom.

For TSB and TBW defined VA spaces that are <512mbytes, JIO IOMMU detects “Out of Range Virtual Address Error”and logs it as ERR_BAD_VA in IOMMU Control Register. Those addresses get rejected by the IOMMU and for a read access target abort gets issued on the PCI bus.

For <512 MB TSB and TBW defined VA space size with multiple 512 MB segments enabled, addresses in the different segments defined as legal by the TSB and TBW size can alias to the same TSB entry. Illegal addresses get “Out of Range Virtual Address Error” .

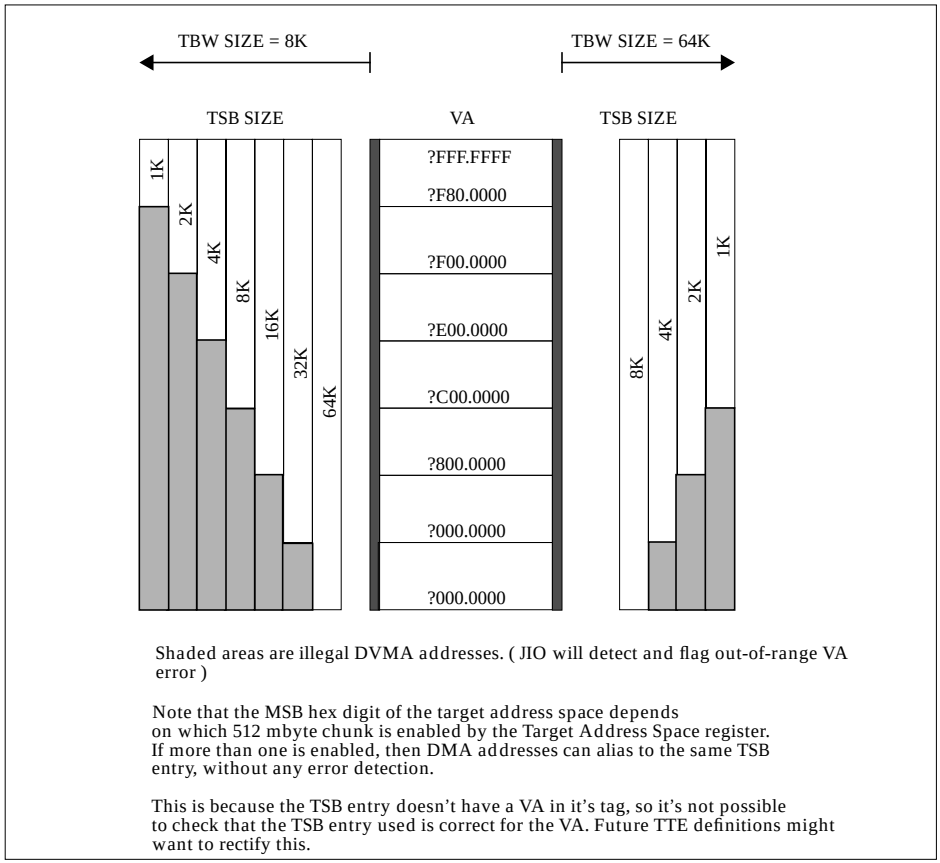
But for TSB and TBW defined VA space sizes that are 512 MB or greater, JIO does not detect any out of range errors, and aliasing in the TSB always happens.There are only 3 combinations of TBW_SIZE and TSB_SIZE that cause the TSB and TBW defined supported VA space to be larger than 512mbytes (Fig 4-5) .

Illegal Combination of TSB_SIZE and TBW_SIZE Error

The IOMMU also returns an error if it detects illegal combination of TSB_SIZE and TBW_SIZE (see table 4-60). Hardware does not prevent illegal combinations from being programmed. If an illegal combination is programmed into the IOMMU, all translation requests will be rejected as invalid

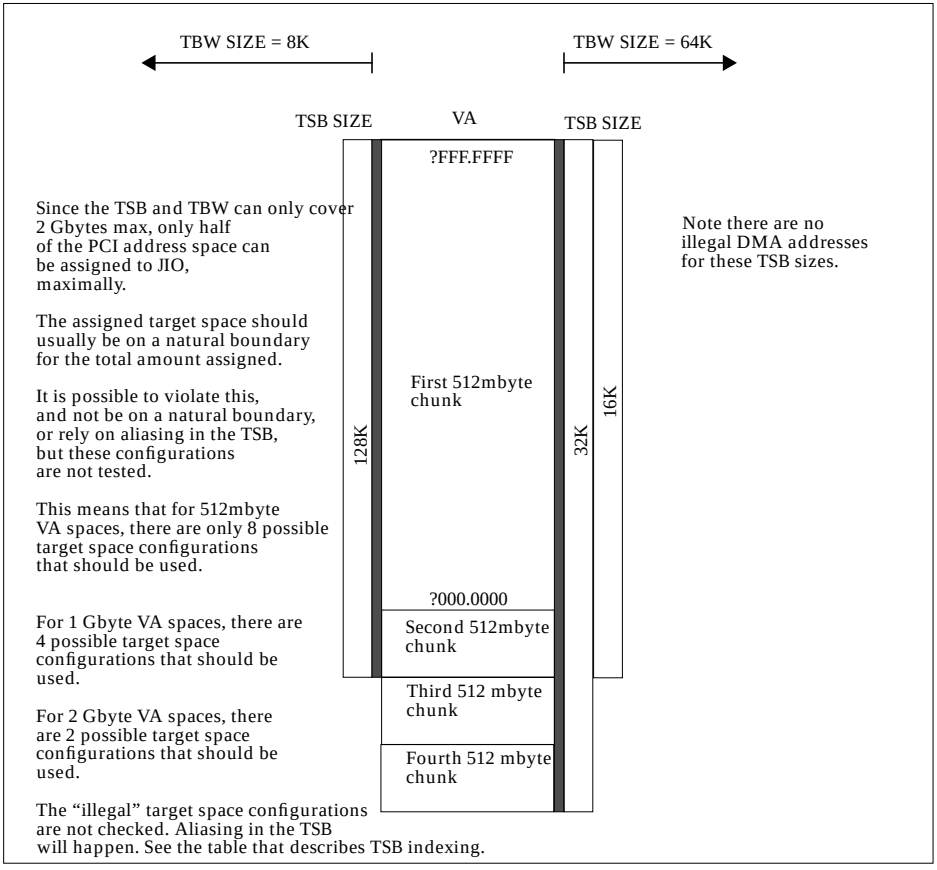
The error gets logged in ERR_ILLEGAL_TSB_TBW" field in the IOMMU Control Register and the ERR bit gets set.

FIGURE 4-4 Legal DVMA Address Configurations when Target Space is <512mbytes.



There are no illegal cases detected for TSB/TBW sizes that create > 512mbytes of DVMA space. Specifically, the 128k entry, 8Kbyte page size. And the 16k and 32k entries for 64Kbyte page size.

FIGURE 4-5 Legal DVMA Address Configurations when Target Space is 512,1024, or 2048 mbytes.



Supported Target Address Space Definitions

Since the TSB organization has a contiguous index range as shown in Table 4-60, the 512 Mbyte chunks in the PCI Target Address Space Register should be assigned contigously, and preferably on the natural boundary of the TSB size irrespective of whether the total target space assigned is the same or bigger than the TSB size.

However it is possible to violate this and JIO does support 3 cases of misaligned 1GB DVMA target space programming (1GB TSB and TBW defined VA space size) and 3 cases of misaligned 2GB DVMA target space programming (2GB TSB and TBW defined VA space size) as shown in Fig 4-6.

For the 1GB misaligned programming cases with 1GB TSB and TBW defined VA space , IOMMU logic decodes SEG_DISP [1:0] == 01 in IOMMU Control Register and flips VA[29] to map the linear Target Address Space correctly to the linear TSB range, thereby avoiding the

FIGURE 4-6 Supported Target Address Space Definitions

Target Address Space Register [7:0]	SEG_DISP[1:0]	TSB / TBW defined VA space	IOMMU Hardware Action
1000. 0000 0100. 0000 0010. 0000 0001. 0000 0000. 1000 0000. 0100 0000. 0010 0000. 0001	XX	< 512 MB 512 MB	-Out of Range Error for illegal addresses for < 512 MB VA space - All addresses are legal for 512 MB VA space and accepted by IOMMU - No modification to VA[30:29]
1100. 0000 0011. 0000 0000. 1100 0000. 0011	XX XX 00	< 512 MB 512 MB 1GB	-Out of Range Error for illegal addresses for < 512 MB VA space - Legal addresses for <512 MB VA space sizes alias. - Aliasing for 512 MB VA spaces - All addresses legal for 512 MB and 1GB VA spaces. -No modification to VA[30:29]
1111. 0000 0000. 1111	XX XX 00 00	< 512 MB 512 MB 1GB 2GB	-Out of Range Error for illegal addresses for < 512 MB VA spaces - Legal addresses for <512 MB VA spaces alias - Aliasing for 512 MB & 1GB VA - All addresses legal for 512 MB, 1GB and 2 GB VA spaces -No modification to VA[30:29]
0110.0000 0001.1000 0000.0110	01	1GB (other VA space sizes not shown)	- Invert VA[29]. VA[30] unchanged
0011.1100 0111.1000 0001.1110	10 01 11	2GB 2GB 2GB (other VA space sizes not shown)	- Invert VA[30]. VA[29] unchanged - Increment VA[30:29] by 1 - Decrement VA[30:29] by 1

adverse effect of having the high DVMA addresses (within the 1GB) ending up using lower half of the TSB and low DVMA addresses (within the 1GB) ending up using upper half of the TSB.

For the 2GB misaligned programming cases with 2 GB TSB and TBW defined VA space , IOMMU logic decodes SEG_DISP[1:0] == 01/10/11 to increment VA[30:29] by 1 /flip VA[30] /decrement VA[30:29] by 1 respectively, to map the linear Target Address Space correctly to the linear TSB range.

Note – IOMMU logic can implement desired VA[30:29] manipulations by using a 2-bit adder to add SEG_DISP[1:0] bits to VA[30:29].

The assumption is software would always correctly program SEG_DISP[1:0] corresponding to the programming of the PCI Target Address Space Register.

Translation-Lookaside Buffer (TLB) Locking

For diagnostics and debugging, the IOMMU has the capability of restricting itself to use just a single entry of the TLB. This is controlled by the LRU_LCKEN and LRU_LCKPTR fields of the IOMMU Control Register. To properly turn locking on the following sequence is required:

- Set MMU_EN to 0
- Set LRU_LCKEN to 1 (must be a separate PIO write)
- Set LRU_LCKPTR to desired value (may be combined with previous PIO)
- Set MME_DE to 1 (may be combined with previous PIO)
- Invalidate all TLB entries
- Set MMU_EN to 1 and MMU_DE to 0.

To unlock the TLB:

- Set LRU_LCKEN to 0

4.5.3.3 TSB Base Address Register

The TSB Base Address Register contains the pointer to the first-entry of the TSB table. Together with part of the virtual address it uniquely identifies the address where hardware should fetch the TTE from the TSB table. The TSB table has to be

aligned on 8K boundary. The lower order 13 bits are assumed to be 0x0 during TSB table lookup. Tables larger than 8K bytes are only constrained to be on 8K boundaries rather than having to be size aligned.

TABLE 4-61 TSB Base Address Register(PCI_CSRBase+0x00.0208)

Field	Bits	Reset	Description	Type
RESERVED	63:43	0	Reserved, read as zeros	R
TSB_BASE	42:13	X	Upper 30bits of the PCI TSBs physical address	R/W
RESERVED	12:0	0	Reserved, read as zeros	R

4.5.3.4 Flush Page Register

This is a write-only register that allows software to perform address based flushes of a mapping from TLB. The data written to this address contains the page number to be flushed. If there is a TLB entry with a matching page number, it will be invalidated.

TABLE 4-62 Flush Page Register (PCI_CSRBase+0x00.0210)

Field	Bits	Reset	Description	Type
Reserved	63:32	n/a	Reserved, write has no effect	W
FLUSH_VPN	31:13	n/a	31:16 = virtual page number if 64K page; bits 15:13 are don't care 31:13 = virtual page number if 8K page	W
Reserved	12:0	n/a	Reserved, write has no effect	W

Note – No hardware mechanisms exist to solve the potential race between a DVMA translation needing a TLB entry and the write to the Flush Page Register intended to flush that entry. Software must manage the interlock by guaranteeing that no DVMA can be going on to the page which is being flushed.

4.5.3.5 Flush Context Register

This is a write-only register that allows software to perform context based flushes of multiple mappings from TLB. The data written to this address contains the context number to be flushed. If there are any TLB entries with matching context numbers, they will be invalidated.

TABLE 4-63 Flush Context Register (PCI_CSRBase+0x00.0218)

Field	Bits	Reset	Description	Type
Reserved	63:12	n/a	Reserved, write has no effect	W
FLUSH_CONTEXT	11:0	n/a	Context number to flush. Context numbers have no inherent meaning to the IOMMU, but are used by software to group related pages.	W

Note – No hardware mechanisms exist to solve the potential race between a DVMA translation needing a TLB entry and the write to the Flush Context Register intended to flush that entry. Software must manage the interlock by guaranteeing that no DVMA can be going on to the context which is being flushed.

4.5.3.6 Translation Fault Address Register

This register is not present in SUPCI. It is provided to aid debug of software errors. It logs the PA of the TTE entry on Invalid, Protection (IOM miss) errors. If the protection error had an IOM hit, the translated PA from the IOM is saved instead of the PA of the TTE entry.This may occur if a prior DMA read caused the IOM entry to be installed.

Note that UE error associated with TTE data gets logged along with the PA in the Jbus UE AFSR and AFAR registers. In Sabre it used to be logged in a separate DMA UE AFSR/AFAR. JIO is software compatible with SUPCI, and SUPCI logs the UE,CE for TTE data in the Safari UE AFAR and AFSR registers.

To find out the PA for an IOM error, software needs to probe the Jbus UE AFSR and AFAR registers for UE related information, and this register for the translation error (Invalid and Protection Error) PAs.

Note – This register gets updated when Invalid or Protection (IOM miss) error gets detected by JIO IOMMU while ERR bit [bit 24] in IOMMU control Register = 1'b0. In other words, this register gets updated only on the occurrence of the first IOMMU error (indicated by ERR bit in IOMMU control register being 1'b0) of type Invalid or Protection. Once updated due to an error, it would not be later possible to tell if one error or multiple errors of the same type occurred, but the address is guaranteed to be of the first error.

Note – Note that the reset value of this register is 0x0. This should be fine provided software looks at this register only after the occurance of the error, and the system does not get reset between the time the error occurs and software examines the contents of this register.

TABLE 4-64 Translation Fault Address Register (PCI_CSRBase+0x00.0220)

Field	Bits	Reset	Description	Type
Reserved	63:43	0x0	Reserved.Read as zeros.	R
Translation Error PA	42:0	0x00	PA for IOM translation error (Invalid or Protection).	R

4.5.3.7 TLB TAG Diagnostics Access

The TLB Tag Diagnostics Access provides diagnostics path to the 16-entry TLB Tag when the MMU_DE bit in the IOMMU Control Register is turned on.

TABLE 4-65 TLB Tag Diagnostics Access Regs (PCI_CSRBase+{0x00.A580 - 0x00.A5FF})

Field	Bits	Reset	Description	Type
RESERVED	63:37	0	Reserved, read as zeros	R
CONTEXT	36:25	X	Context Number	R/W
ERRSTS	24:23	X	Error Status: 00 = Protection Error 01 = Invalid Error 10 = Timeout 11 = ECC Error (UE)	R/W
ERR	22	X	When set to 1, indicates that there is an error associated with this TLB entry. The specific error is indicated by the ERRSTS field.	R/W

TABLE 4-65 TLB Tag Diagnostics Access Regs (PCI_CSRBase+{0x00.A580 - 0x00.A5FF})

Field	Bits	Reset	Description	Type
W	21	X	Writable bit. when set, the page mapped by the TLB has write permission granted.	R/W
Rsvd	20	0	Reserved.Read as zero.	R
SIZE	19	X	Page Size, 0=8K and 1=64K.	R/W
VPN	18:0	X	VPN[31:13]	R/W

Note – Diagnostic accesses should insure that multiple page match conditions are not generated. The result of multiple page matches is unpredictable (multiple context matches are allowed).

4.5.3.8 TLB Data RAM Diagnostic Access

The TLB Data Diagnostics Access provides direct PIO accesses to 16 entries of TLB Data RAM. MMU_DE bit in the IOMMU Control Register must be turned on to perform the accesses. Following table shows the information included in the returned data.

TABLE 4-66 TLB Data RAM Diagnostics Access Regs (PCI_CSRBase+ {0x00.A600 - 0x00.A67F})

Field	Bits	Reset	Description	Type
RESERVED	63:33	0	Reserved, read as 0.	R
V	32	0	Valid bit, when set, the TLB data field is meaningful.	R/W
RESERVED	31	0	Reserved, read as 0 (was local bus bit for SBus)	R/W
C	30	X	Cacheable bit. 1=Cacheable access, 0=Non-cacheable.	R/W
PA[42:13]	29:0	X	30-bit Physical Page Number.	R/W

4.5.3.9 TLB Compare Setup Diagnostic Register

This register is used to set up the virtual address or context number for TLB compare diagnostics. The virtual address or context number is written to this register along with a flag indicating which one should be used, and the compare results from TLB can be read from the TLB Compare Result Diag Register.

TABLE 4-67 TLB Compare Setup Diagnostic Register(PCI_CSRBase+0x00.A400)

Field	Bits	Reset	Description	Type
RESERVED	63:33	0	Reserved, read as 0.	R
MATCH_TYPE	32	X	Set to 0 to select lookup by virtual page number, or 1 to select lookup by context number.	R/W
VPN	31:13	X	Virtual page number.	R/W
RESERVED	12	0	Reserved, read as 0.	R
CONTEXT	11:0	X	Context number for compare.	R/W

4.5.3.10 TLB Compare Result Diagnostic Access

TABLE 4-68 TLB Tag Comparator Result Diagnostic Register (PCI_CSRBase+0x00.A408)

Field	Bits	Reset	Description	Type
RESERVED	63:16	0	Reserved, read as zeros	R
COMP	15:0	X	TLB tag comparator output for each entry.	R

Note – The TLB Tag Comparator Diagnostics Access provides diagnostics path to the 16-entry TLB Tag Comparator when the MMU_DE bit in the IOMMU Control Register is turned on. Bit 0 represents the comparison result of the first TLB Tag entry, and bit 15 represents the last.

In order to avoid invalid address translation after TLB diagnostics, the valid bits in the TLB should be reset appropriately before doing any meaningful address translation.

4.5.4 Streaming Cache Registers

TABLE 4-69 Offset of Streaming Cache Registers

Register	Address	Access Size
Streaming Cache Control Reg.	PCI_CSRBase+0x00.2800	8 bytes
Streaming Cache Page Flush / Invalidate Reg	PCI_CSRBase+0x00.2808	8 bytes
Streaming Cache Flush Synchronization Reg	PCI_CSRBase+0x00.2810	8 bytes
Streaming Cache Context Flush / Invalidate Reg	PCI_CSRBase+0x00.2818	8 bytes
Streaming Cache Data RAM Diagnostic	PCI_CSRBase+0x00.B000 - PCI_CSRBase+0x00.B7FF	8 bytes
Streaming Cache Error Status Diagnostics	PCI_CSRBase+0x00.B800 - PCI_CSRBase+0x00.B8FF	8 bytes
Streaming Cache Page Tag Diagnostics	PCI_CSRBase+0x00.BA00 - PCI_CSRBase+0x00.BA7F	8 bytes
Streaming Cache Line Tag Diagnostics	PCI_CSRBase+0x00.BB00 - PCI_CSRBase+0x00.BB7F	8 bytes
Streaming Cache Context Match Reg	PCI_CSRBase+0x01.0000 - PCI_CSRBase+0x01.7FFF	8 bytes

Note that all the streaming cache registers are reserved for JIO. Reads to these registers would simply return undefined data.

4.5.5 Interrupt Registers

PCI related interrupts are delivered to the processor by JIO as a 5-cycle packet by way of the Jbus interrupt transaction type, and have the format shown in TABLE 4-70.

TABLE 4-70 Jbus interrupt packet format

Interrupt packet J_AD, J_ADTYPE	Bits	Definition
1st Cycle : J_AD	30:0	Reserved, issued as 0.
	35:31	JPID of Source Port . Will be {JPID[4:1],1'b0} for PCI and {JPID[4:1],1'b1} for GR
	40:36	JPID of Target Port
	42:41	Reserved, issued as 0.
	47:43	0x14, Jbus Transaction Type = INT
	63:48	Reserved, issued as 0.
	127:64	Same as 63:0
1st Cycle : J_ADTYPE	1:0	Reserved, issued as 0.
	5:2	Src ID (Jbus Port ID[3:0] of src). If JIO is src, then interrupt can be either from PCI (src_id = JIO JPID[3:0]) or from Graphics (src_id = {JPID[3:1],1'b1}).
	7:6	Address Cycle. Issued as 1's.
2nd Cycle : J_AD	63 :0	Reserved, issued as 0's
	69:64	Interrupt Number Offset (INO)
	74:70	Interrupt Group Number (IGN) . Same as Bits 35:31 on J_AD in first cycle.
	127:75	Reserved, issued as 0's
2nd Cycle : J_ADTYPE	7:0	8'b0
3rd Cycle : J_AD	127:0	Reserved, issued as 0's
3rd Cycle : J_ADTYPE	7:0	8'b0
4th Cycle : J_AD	127:0	Reserved, issued as 0s

TABLE 4-70 Jbus interrupt packet format

Interrupt packet J_AD, J_ADTYPE	Bits	Definition
4th Cycle : J_ADTYPE	7:0	8'b0
5th Cycle : J_AD	127:0	Reserved, issued as 0s
5th Cycle : J_ADTYPE	7:0	8'b0

Note – The Mondo Vector Format has been changed from v3.0 of the PRM to match the Format specified by Sparc IIIi system specs. This change is indicated in Revision History section of the PRM (Chapter 1 : JIO Basics).

The IGN and INO field in the first doubleword of the interrupt data packet together make up an 11 - bit interrupt number (INR), which indicates the source of the interrupt. Where possible, the interrupt is precise (i.e., it points to only one interrupt source). This permits the dispatch of the proper interrupt service routine without any register polling. Above the first doubleword the bits of the packet are guaranteed to be zero. Software can use this knowledge to distinguish these interrupts from others such as cross-calls.

JIO would support only one Interrupt Controller in the Primary PCI Leaf. To stay software compatible however, the two sets of logical CSR addresses for the 2 interrupt controllers in SUPCI would alias to one set of physical registers in the I-chip Node. Thus either of the 2 CSR address ranges (PCI-A_CSRBase, PCI-B_CSRBase) can be used to access one set of physical registers.The only exception is that references to the PCI Consistent DMA Flush/Sync Registers for the 2 PCI domains are treated as unmapped.

In addition to PCI related interrupts, the primary PCI leaf within JIO is responsible for generating some interrupts on behalf of other portions of the chip, including some internal sources (uncorrectable and correctable ECC errors) as well as external sources (UPA64S port interrupts).

Note – JIO Interrupt unit processes 56 interrupts. 48 of them are received from external Ichip in encoded format and 8 of them are internal from various blocks. Out of 48 external interrupts,32 of them are PCI related interrupts (interrupts generated from external PCI agents), 14 On board IO and 2 UPA related interrupts.

Note – All interrupts processed by JIO are dispatched after dispatching current and previously processed DMA writes on to the Jbus. This is known as hardware DMA synchronization. The future DMA transactions are retried.The hardware DMA synchronization and dispatch of interrupts occurs for PCI A Bus or PCI B bus of JIO according to the value of the corresponding bit in the Interrupt Routing Register (0 : PCI A, 1 : PCI B). In cases of DMA writes coming from downstream bridges, it may be necessary to flush those writes even after the interrupt delivery. In that case, software would flush those remaining writes by software DMA synchronization (please refer to section 4.5.7.1)

Note – Hardware DMA synchronization preserves Interrupt/store order because the CPU that receives the interrupt does not trap untill all pending snoops and invalidates (issued from JIO as WRIS transactions) in its input queues are processed. However when software does a software DMA synchronization in JIO by issuing a PIO store followed by a PIO load polling on the synchronization status, there is nothing in the CPU to force completion of its queued snoops and invalidates before it can use the PIO read data from JIO indicating synchronization is complete in JIO. Thus it is possible for CPU to read the status from JIO indicating that the sync is done , but there are still pending invalidates in its queue that are not processed yet. To make sure that these invalidates are processed before CPU mistakenly reads old data from memory (thinking it is new data written by JIO and thereby cause data corruption), the sync code should introduce a Bstore Commit / Membar #sync right after the PIO load . The Bstore Commit guarantees invalidate completion to itself, and since invalidates complete in order, so it has the effect of completing all pending invalidates in itself as well as other CPU's.

There are 64 available INO values available to JIO, which are broken down into groups according to TABLE 4-71.

TABLE 4-71 Interrupt Number Offset Assignments

INO (binary)	INO (hex)	Interrupt Source
0sssnn	00-1f	PCI Slot Interrupts, 4 separate interrupts per slot. sss = PCI slot number, 0-7 nn = 00 for INTA#, 01 for INTB#, nn = 10 for INTC#, 11 for INTD#
10nnnn	20-2f	On-board I/O (OBIO) interrupts

TABLE 4-71 Interrupt Number Offset Assignments

INO (binary)	INO (hex)	Interrupt Source
10101n	2a, 2b	UPA slot interrupts (included in OBIO category) Pulse type interrupts
110nnn	30-37	Internal interrupts, where nnn is defined as: 000 = UE (Uncorrectable ECC error) 001 = CE (Correctable ECC error) 010 = AE (PCI Bus A Error) 011 = BE (PCI Bus B Error) 100 = SE (Jbus Errors) 101, 110, 111 = Reserved
111nnn	38-3e	Reserved for JIO.

Note – All of the internal interrupts in JIO as shown in Table 4-71 are CSR driven. In other words the interrupt enables for these internal errors merely delay the interrupt reporting. If an error happens but the interrupt enable is off, the interrupt does not happen . But enabling the interrupt later on, with the CSR bit inidicating the error still set, would cause the interrupt to happen. For each INO ranging from 30 to 34 in Table 4-71, software should clear all error bits covered by that INO , while servicing an internal error interrupt for that INO.

The following figure shows the different interrupts that can be serviced by JIO along with their respective INO's, sources,type and priorities :

FIGURE 4-7 Summary of Interrupts

Interrupt	Int/Ext	Source	INO(hex)	Type	INO (bin)	Priority
Reset Condition	Ext		0x3F	Level		
PCI_BUS0_INT_L[0]	Ext	PCI	0x00	Level	000000	1
PCI_BUS0_INT_L[1]	Ext	PCI	0x01	Level	000001	2
PCI_BUS0_INT_L[2]	Ext	PCI	0x02	Level	000010	3
PCI_BUS0_INT_L[3]	Ext	PCI	0x03	Level	000011	4
PCI_BUS1_INT_L[0]	Ext	PCI	0x04	Level	000100	5
PCI_BUS1_INT_L[1]	Ext	PCI	0x05	Level	000101	6
PCI_BUS1_INT_L[2]	Ext	PCI	0x06	Level	000110	7
PCI_BUS1_INT_L[3]	Ext	PCI	0x07	Level	000111	8

Interrupt	Int/Ext	Source	INO(hex)	Type	INO (bin)	Priority
PCI_BUS2_INT_L[0]	Ext	PCI	0x08	Level	001000	1
PCI_BUS2_INT_L[1]	Ext	PCI	0x09	Level	001001	2
PCI_BUS2_INT_L[2]	Ext	PCI	0x0A	Level	001010	3
PCI_BUS2_INT_L[3]	Ext	PCI	0x0B	Level	001011	4
PCI_BUS3_INT_L[0]	Ext	PCI	0x0C	Level	001100	5
PCI_BUS3_INT_L[1]	Ext	PCI	0x0D	Level	001101	6
PCI_BUS3_INT_L[2]	Ext	PCI	0x0E	Level	001110	7
PCI_BUS3_INT_L[3]	Ext	PCI	0x0F	Level	001111	8
PCI_BUS4_INT_L[0]	Ext	PCI	0x10	Level	010000	1
PCI_BUS4_INT_L[1]	Ext	PCI	0x11	Level	010001	2
PCI_BUS4_INT_L[2]	Ext	PCI	0x12	Level	010010	3
PCI_BUS4_INT_L[3]	Ext	PCI	0x13	Level	010011	4
PCI_BUS5_INT_L[0]	Ext	PCI	0x14	Level	010100	5
PCI_BUS5_INT_L[1]	Ext	PCI	0x15	Level	010101	6
PCI_BUS5_INT_L[2]	Ext	PCI	0x16	Level	010110	7
PCI_BUS5_INT_L[3]	Ext	PCI	0x17	Level	010111	8
PCI_BUS6_INT_L[0]	Ext	PCI	0x18	Level	011000	1
PCI_BUS6_INT_L[1]	Ext	PCI	0x19	Level	011001	2
PCI_BUS6_INT_L[2]	Ext	PCI	0x1A	Level	011010	3
PCI_BUS6_INT_L[3]	Ext	PCI	0x1B	Level	011011	4
PCI_BUS7_INT_L[0]	Ext	PCI	0x1C	Level	011100	5
PCI_BUS7_INT_L[1]	Ext	PCI	0x1D	Level	011101	6
PCI_BUS7_INT_L[2]	Ext	PCI	0x1E	Level	011110	7
PCI_BUS7_INT_L[3]	Ext	PCI	0x1F	Level	011111	8
OBIO0_INT_L[0]	Ext	OBIO	0x20	Level	100000	1
OBIO0_INT_L[1]	Ext	OBIO	0x21	Level	100001	2
OBIO0_INT_L[2]	Ext	OBIO	0x22	Level	100010	3
OBIO0_INT_L[3]	Ext	OBIO	0x23	Level	100011	4
OBIO0_INT_L[4]	Ext	OBIO	0x24	Level	100100	5
OBIO0_INT_L[5]	Ext	OBIO	0x25	Level	100101	6

Interrupt	Int/Ext	Source	INO(hex)	Type	INO (bin)	Priority
OBIO0_INT_L[6]	Ext	OBIO	0x26	Level	100110	7
OBIO0_INT_L[7]	Ext	OBIO	0x27	Level	100111	8
OBIO0_INT_L[8]	Ext	OBIO	0x28	Level	101000	1
OBIO0_INT_L[9]	Ext	OBIO	0x29	Level	101001	2
UPA_INT_L[0]	Ext	OBIO	0x2A	Pulse	101010	3
UPA_INT_L[1]	Ext	OBIO	0x2B	Pulse	101011	4
OBIO1_INT_L[0]	Ext	OBIO	0x2C	Level	101100	5
OBIO1_INT_L[1]	Ext	OBIO	0x2D	Level	101101	6
OBIO1_INT_L[2]	Ext	OBIO	0x2E	Level	101110	7
OBIO1_INT_L[3]	Ext	OBIO	0x2F	Level	101111	8
Uncorrectable error	Int	DRAM	0x30	Level	111000	1
correctable error	Int	DRAM	0x31	Level	111001	2
PCI Bus A error	Int	PCI	0x32	Level	111010	3
PCI Bus B error	Int	PCI	0x33	Level	111011	4
Jbus Error	Int	JBus	0x34	Level	111100	5
Reserved	Int		0x35	Level	111101	6
Reserved	Int		0x36	Level	111110	7
Reserved	Int		0x37	Level	111111	8

The primary PCI leaf has a set of registers for all of the possible interrupt sources.These registers are listed in TABLE 4-72

TABLE 4-72 Offset of Interrupt Registers

Register	Offset	Access Size
Interrupt Mapping Register for interrupt INO	PCI_CSRBase+ 0x00.1000+INO*8	8 bytes
UPA Port 0 Interrupt Mapping Register (register is also mapped normally via INO)	PCI_CSRBase+ 0x00.6000	8 bytes
UPA Port 1 Interrupt Mapping Register (register is also mapped normally via INO)	PCI_CSRBase+ 0x00.8000	8 bytes
Clear Interrupt Register for interrupt INO	PCI_CSRBase+ 0x00.1400+INO*8	8 bytes
PCI Int State Diag Register	PCI-_CSRBase+0x00.A800	8 bytes

TABLE 4-72 Offset of Interrupt Registers

Register	Offset	Access Size
OBIO and Internl Int State Diag Register	PCI_CSRBase+0x00.A808	8 bytes
Interrupt Retry Register	PCI_CSRBase+0x00.1A00	8 bytes
PCI Consitent DMA Flush/Sync Register	PCI_CSRBase+0x00.1A08	8 bytes

4.5.5.1 Interrupt Mapping Registers

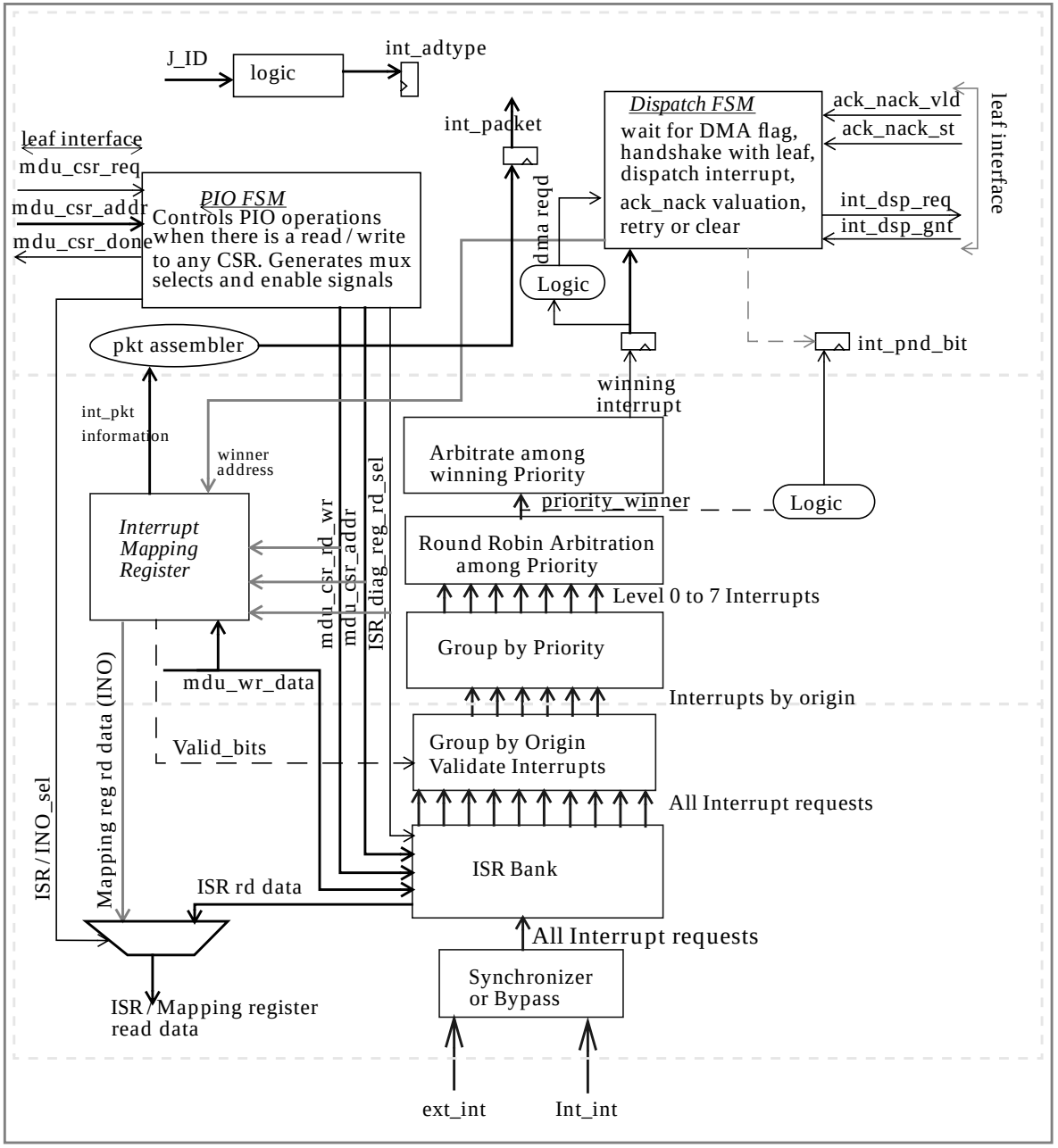
The format for each interrupt mapping registers is shown in TABLE 4-73

TABLE 4-73 Format of Interrupt Mapping Registers (PCI_CSRBase+0x00.1000+INO*8)

Field	Bits	Reset	Description	Type
Reserved	63:32	0	Reserved, read as 0	R
V	31	0	Valid bit When set to 0, interrupt will not be dispatched to CPU from this PCI leaf. Has no other impact on interrupt state.	R/W
T_JPID	30:26	X	JPID of the processor that this interrupt will be sent to.	R/W
Reserved	25:21	0	Reserved, read as 0	R
Reserved	20:11	0	Reserved, read as 0	R
IGN	10:6	-	Interrupt Group Number. This will be JPID[4:0] where JPID(JIO's JPID(Jbus Port ID) is defined in JIO Control/Status Register.	R
INO	5:0	-	Interrupt Number Offset The value of this field is hardwired for each mapping register, as shown in TABLE 4-72	R

The valid bits are looked at when the Interrupt requests are recorded and the requests are forwarded to the arbiter. Please refer to the figure below for detailed description of how an interrupt gets processed within JIO.

Figure 4-8 Interrupt Processing in JIO



Interrupt gets processed in JIO in the following steps :

1) The interrupt requests are recorded in the ISRs (Interrupt State Diagnostic Registers)

2)When the ISRs = RECEIVE state and corresponding valid bit is on (= 1) then the requests are considered valid

3) Once a valid requests are seen, the first level of arbitration is done. In the first level of arbitration, all the 56 interrupts are categorized into 8 prorities. So the output of the first level of arbitration is a prority level (which has the request present in it).

4) The output of the first arbiter will select one of the 7 interrupt request groups. These requests are fed to the second level of arbitration .

5) The output of the second level of arbitration is the winning interrupt.

6) Once the winner is determined, the winner valid signal is also asserted notifying the logic to process this winning interrupt.

7)The winner is registered and dispatch happens .

8) Once the dispatch is completed upon receiving INTACK from CPU, the current grant is transfered to last grant and new requests are processed. If an INTNACK is received, the interrupt is redispatched after a back off time programmed into the retry register. The valid bits have no impact while the interrupt is being dispatched.

4.5.5.2 Interrupt Clear Registers

Each interrupt source also has a state register associated with it. This state register can be either of type “level” or of type “pulse.”

In the level sensitive case, the state register has two bits, and there are three valid states: IDLE, RECEIVED, and PENDING. IDLE represents the state where no interrupts are reported. RECEIVED indicates that an interrupt has been detected and should be delivered to the processor if/when the valid bit is set in its the mapping register. PENDING is the state when the interrupt has been sent to the processor. Any subsequent detection of the same interrupt is ignored until software resets the state machine back to IDLE.

The state register for each level sensitive interrupts can be set to any desired state by software via the Clear Interrupt Registers.

Note – Software should ensure that the source of a level sensitive interrupt (i.e. the PCI device, or JIO internal register) is cleared before clearing the interrupt state register via the Interrupt Clear Register, otherwise JIO will continue to reissue the interrupt each time the state register is set to IDLE.

In the pulse case, the state register allows only two states: IDLE and RECEIVED. These states have the same meaning as for the level sensitive case. There is no PENDING state, so the state machine transitions from RECEIVED back to IDLE when the interrupt is dispatched to a processor. The only pulse type interrupts are the two UPA port interrupts.

The format for the Interrupt Clear Registers is shown in TABLE 4-74 and TABLE 4-75.

TABLE 4-74 Clear Interrupt Register - Level Interrupts(PCI_CSRBase+0x00.1400+INO*8)

Field	Bits	Reset	Description	Type
reserved	63:2	n/a	Reserved. Write with 0.	W
State	1:0	n/a	State bits for the interrupt state machine associated with this interrupt. The following values may be written: 00 - Set state machine to IDLE state 01 - Set state machine to RECEIVED state 10 - Reserved 11 - Set state machine to PENDING state	W

TABLE 4-75 Clear Interrupt Register - Pulse Interrupts(PCI_CSRBase+0x00.1400+INO*8)

Field	Bits	Reset	Description	Type
reserved	63:2	n/a	Reserved. Write with 0.	W
State	1:0	n/a	State bit for the interrupt state machine associated with this interrupt. The following values may be written: 00 - Set state machine to IDLE state 01 - Set state machine to RECEIVED state 10 - Reserved 11 - Reserved	W

Note – The Interrupt Clear Registers are write only. To determine the current interrupt state, use the interrupt state diagnostic registers instead.

4.5.5.3 Interrupt State Diagnostic Registers

The state bits for every interrupt source are made available in one of the two Interrupt State Diagnostic Registers. The state bits here have the same definitions as in TABLE 4-74.

The locations of each set of state bits is derived from the associated INO:

Register: if (INO >= 0x20) then OBIO Int Diag Reg else PCI Int Diag Reg

Bits: Int Diag Reg [(2 * (INO & 0x1F)) + 1 : (2 * (INO & 0x1F))]

TABLE 4-76 PCI Interrupt State Diagnostic Register(PCI-_CSRBase+0x00.A800)

Field	Bits	Reset	Description	Type
PCI_INT_STATE	63:0	0xFFFFFFFF.FFFFFFFF	State bits for PCI interrupts (INO 0x00-0x1f)	R

TABLE 4-77 OBIO and Internal Interrupt State Diagnostic Register (PCI_CSRBase+0x00.A808)

Field	Bits	Reset	Description	Type
Reserved	63:48	0x0000	Reserved, read as 0. INOs reserved for NewLink	R
INT_INT_STATE	47:32	0xFFFF	State bits for Internal interrupts (INO 0x30-0x37).	R
OBIO_INT_STATE	31:0	0xFF0FFFFF	State bits for on-board I/O interrupts (INO 0x20-0x2f).	R

4.5.5.4 Interrupt Retry Timer Register

If an interrupt packet sent by JIO is NACK'd by the Jbus target agent, JIO will wait for a programmed number of clocks and resend the interrupt. This register controls the number of clocks the interrupt dispatch unit should wait before resending the interrupt packet. The count specified by this register is not precise: there is a free running down counter (operating at the 133 MHz/UPA frequency) which is loaded from this value and which must roll through 0 twice before the packet is retried.

TABLE 4-78 Interrupt Retry Timer Register(PCI_CSRBase+0x00.1A00)

Field	Bits	Reset	Description	Type
RESERVED	63:20	0	Reserved, read as 0	R
LIMIT	19:0	0xFFFFF	Limit - the retry interval	R/W

Note – To be compatible with SUPCI with regard to Interrupt re-issue latency, this counter should be incremented at every other 133mhz/UPA clock, assuming the LIMIT value stays the same as SUPCISUPCI.

4.5.6 PCI Performance Monitor Registers

TABLE 4-79 Offset of PCI Performance Monitor Registers

Register	Offset	Access Size
PCI Performance Monitor Control Register	PCI_CSRBase+0x00.0100	8 bytes
PCI Performance Counter Register	PCI_CSRBase+0x00.0108	8 bytes
PCI Idle Check Diagnostics Register	PCI_CSRBase+0x00.0110	8 bytes

In order to gather useful statistics on the PCI performance of JIO, a pair of registers provide counts of key events. There are only two counters present, and the control register selects the input for each of the counters.

In addition, a diagnostic register that indicates whether various subsystems within the PCI leaf are idle is available.

4.5.6.1 Performance Monitor Control Register

This register controls the events to be monitored by the Performance Counter Register.

TABLE 4-80 PCI Performance Monitor Control Register(PCI_CSRBase+0x00.0100)

Field	Bits	Reset	Description	Type
Reserved	63:16	0	Reserved, read as 0	R
SEL1	15:11	0	Select event source for counter 1. See TABLE 4-81 for encoding.	R/W
Reserved	10:9	0	Reserved, read as 0.	R
SEL0	8:4	0	Select event source for counter 0. See TABLE 4-81 for encoding.	R/W
Reserved	3:0	0	Reserved, read as 0	R

The table below defines the encoding for selecting PCI events to monitor in JIO.

TABLE 4-81 PCI Performance Counter Event Sources

SEL0, SEL1	Event Sources
0x00	Reserved
0x01	Reserved
0x02	Number of DVMA read transfers (PCI data beats)
0x03	Number of DVMA write transfers (PCI data beats)
0x04	Reserved
0x05	Number of bus cycles PCI bus is busy with DMA.
0x06	Number of words transferred using DMA on PCI bus A[1]
0x07	Number of bus cycles PCI is busy with PIO.
0x08-0x0f	Reserved
0x10	Number of TLB misses
0x11	Number of interrupts issued from JIO (only logged in the PCI-A side). Software should not program this value as Sel0 or Sel1 in the PCI-B Performance Monitor Control Register.
0x12	Number of interrupt NACKs received by JIO (only logged in the PCI-A side).Software should not program this value as Sel0 or Sel1 in the PCI-B Performance Monitor Control Register.
0x13	Number of PIO read transfers (PCI data beats)
0x14	Number of PIO write transfers (PCI data beats)
0x15	Number of partial line DMA write transactions
0x16	Number of hits in I/O cache due to PCI DMA accesses.
0x17	Reserved.
0x18	Reserved.
0x19	Reserved.
0x1a	Reserved.
0x1b	Reserved.
0x1a-0x1f	Reserved. Counter value is undefined when these sources are chosen.

1. The word count will increment whenever any of the four associated byte enables is active. If, during the recording interval, there are any DMAs that start or end on unaligned addresses, it won't be possible to determine the exact number of bytes transferred.

4.5.6.2 PCI Performance Counter Register

This register contains the counts of the two events selected by the PCI Performance Monitor Control Register. The two counters operate independently, but must both be written together. When either counter reaches its maximum count, it will silently wrap around to 0x0 and continues counting. If necessary, software must detect and handle this overflow condition.

TABLE 4-82 PCI Performance Counter Register(PCI_CSRBase+0x00.0108)

Field	Bits	Reset	Description	Type
CNT1<31:0>	63:32	0	Contains value for event counter 1.	R/W
CNT0<31:0>	31:00	0	Contains value for event counter 0.	R/W

4.5.7 I-chip Nexus Registers

JIO implements a new device node representing the I-chip to re-architect the interrupt scheme in to a more manageable layout. Since the current Interrupt registers are all located in the PCI CSR address space, all of those registers are aliased to this new I-chip register bank that represents the I-chip nexus.The only exception is that the PCI Consistent DMA Flush/Sync Registers for the 2 PCI domains are unimplemented and a new set of registers called DMA Flush/Sync Pending and Complete registers are used by software to synchronize on DMA writes on the two PCI buses. The register descriptions are identical to the ones in the Interrupt section, other than the DMA Flush/sync Pending and Complete Registers.

TABLE 4-83 I-chip Nexus Registers

Register	Offset	Access Size
Interrupt Mapping Register for interrupt INO	Ichip_CSRBase+ 0x00.1000+INO*8	8 bytes
UPA Port 0 Interrupt Mapping Register (register is also mapped normally via INO)	Ichip_CSRBase+ 0x00.6000	8 bytes
UPA Port 1Interrupt Mapping Register (register is also mapped normally via INO)	Ichip_CSRBase+ 0x00.8000	8 bytes
Clear Interrupt Register for interrupt INO	Ichip_CSRBase+ 0x00.1400+INO*8	8 bytes
PCI Int State Diag Register	Ichip_CSRBase+0x00.A800	8 bytes
OBIO and Internal Int State Diag Register	Ichip_CSRBase+0x00.A808	8 bytes

Register	Offset	Access Size
Interrupt Retry Register	Ichip_CSRBase+0x00.1A00	8 bytes
I-chip DMA Flush/Sync Complete Register	Ichip_CSRBase+0x00.1A10	8 bytes
I-chip DMA Flush/Sync Pending Register	Ichip_CSRBase+0x00.1A18	8 bytes

4.5.7.1 I-chip PCI DMA Flush/Sync Pending Register

This register consists of 64 independent programmable bits (one per INO), used by software to initiate a flush of DMA write data for each INO out of JIO.

Software can only set all the bits in this register, it cannot clear any bits. Write with zeroes have no effect on the contents of the register.

Initially software can write 1's to any set of bits in this register to initiate a DMA sync. JIO then remembers the bits set and does DMA sync for only those INOs. After the sync starts and until the sync finishes, software can come and write more bits in this register, but those will not get looked at until the current sync is finished. After the current sync gets finished, the originally set bits in this register gets cleared by hardware, and a new sync starts for the newly set bits if any.

Multiple bits can be set at any one time. The flush is started when at least one bit is set, and when complete clears the originally set bit(s) in the register. A subsequent load to the register would return zeroes on those bits indicating all DMA write data for the INOs in question have been flushed out of JIO.

While the flush is in progress, JIO would block all incoming DMA on that PCI bus.

TABLE 4-84 I-chip PCI DMA Flush/Sync Pending Register (Ichip_CSRBase+0x00.1A18)

Field	Bits	Reset	Description	Type
FLUSH_INO	63:0	0	One bit per INO. When set to 1 means pending DRAIN for that INO. Multiple bits can be set to 1 at the same time. Cannot be cleared to zeroes by software. Cleared only by hardware after DMA synchronization is complete.	R/W

JIO logic figures out whether to flush DMA writes on PCI A/PCI B side or both by looking up the Interrupt Routing Register (programmed by OBP and never changed), to see if the INO bits set in the PCI DMA Flush/Sync Pending Register correspond to the PCI A or PCI B sides. It is possible that some INO bits are set corresponding to the PCI A side and some for the PCI B side, in which case JIO logic will flush both sides. If the INO bit(s) that are set correspond to only the PCI A side

(as per the Routing Register Mapping), only the DMA in the PCI A side gets flushed. If the INO bit(s) that are set correspond to only the PCI B side (as per the Routing Register Mapping), only the DMA in the PCI B side gets flushed.

4.5.7.2 I-chip PCI DMA Flush/Sync Complete Register

This register is a copy of the DMA Flush/Sync Pending Register and is used mainly for diagnostic purposes. Software can individually clear each bit in this register by writing a 1 to it. When the originally set bits in the Pending register gets cleared by hardware after flush completion, they are copied into the Complete register, giving software visibility to the initial INO bits for which flush was completed.

TABLE 4-85 I-chip PCI DMA Flush/Sync Complete Register(Ichip_CSRBase+0x00.1A10)

Field	Bits	Reset	Description	Type
FLUSH_INO_DONE	63:0	0	Copy of INO bits in DMA Flush/Sync Pending Register for which Flush have been completed.	R/W1C

Programming Note: The programming steps for doing Software DMA synchronization are:

(i) Set bits corresponding to INO's in the DMA Flush/Sync Pending Register, for which we want to have DMA writes flushed out of JIO.

(ii) Keep reading the DMA Flush/Sync Pending Register until it indicates all zeroes for the set of bits programmed. All zeroes on these bits signify that the DMA synchronization is complete in JIO and all DMA writes for these INOs have been flushed out of JIO.

(iii) Read the contents of the DMA Flush/Sync Complete Register to see the originally programmed INO bits for which the sync was completed.

(iv) Clear the DMA Flush/Sync Complete Register on a bit-by bit basis by writing 1's to them (per INO).

Note – In between steps (i) and (ii) , software can set more bits in the DMA Flush/Sync Register which do not get looked at by JIO until the current sync is complete, and get looked at for the next sync.

4.6 Register Summary

The following table is a summary of all on chip registers in JIO with proper offsets and access sizes.

TABLE 4-86 Summary Table of All Registers in JIO

Register	Address	Access Size
Jbus CSR Space		
Jbus Device ID Register	JBUS_Base + 0x00.0000	8 bytes
UPA0 Offset Base Register	JBUS_Base + 0x40.0000	8 bytes
UPA0 Offset Mask Register	JBUS_Base + 0x40.0008	8 bytes
UPA1 Offset Base Register	JBUS_Base + 0x40.0010	8 bytes
UPA1 Offset Mask Register	JBUS_Base + 0x40.0018	8 bytes
PCI-A_Mem Offset Base Register	JBUS_Base + 0x40.0040	8 bytes
PCI-A_Mem Offset Mask Register	JBUS_Base + 0x40.0048	8 bytes
PCI-A_Cfg_IO Offset Base Register	JBUS_Base + 0x40.0050	8 bytes
PCI-A_Cfg_IO Offset Mask Register	JBUS_Base + 0x40.0058	8 bytes
PCI-B_Mem Offset Base Register	JBUS_Base + 0x40.0060	8 bytes
PCI-B_Mem Offset Mask Register	JBUS_Base + 0x40.0068	8 bytes
PCI-B_Cfg_IO Offset Base Register	JBUS_Base + 0x40.0070	8 bytes
PCI-B_Cfg_IO Offset Mask Register	JBUS_Base + 0x40.0078	8 bytes
JIO Control/Status Register	JBUS_Base + 0x41.0000	8 bytes
Jbus Error Control Register	JBUS_Base + 0x41.0008	8 bytes
Jbus Interrupt Control Register	JBUS_Base + 0x41.0010	8 bytes
Jbus Error Log Register	JBUS_Base + 0x41.0018	8 bytes
Jbus Parity Control Register	JBUS_Base + 0x41.0020	8 bytes
UE AFSR	JBUS_Base + 0x41.0030	8 bytes
UE AFAR	JBUS_Base + 0x41.0038	8 bytes
CE AFSR	JBUS_Base + 0x41.0040	8 bytes
CE AFAR	JBUS_Base + 0x41.0048	8 bytes
Jbus Energy Star Control Register	JBUS_Base + 0x41.0050	8 bytes

Register	Address	Access Size
Jbus Change Initiation Register	JBUS_Base + 0x41.0058	8 bytes
Jbus DTag Diagnostic Registers	JBUS_Base + {0x41.2000 - 0x41.2070}	8 bytes
Jbus CTag Diagnostic Registers	JBUS_Base + {0x41.3000 - 0x41.3070}	8 bytes
Jbus Performance Control Register	JBUS_Base + 0x41.7000	8 bytes
Jbus Performance Counter Register	JBUS_Base + 0x41.7008	8 bytes
Reset_Gen Register	JBUS_Base + 0x41.7010	8 bytes
Reset_Source Register	JBUS_Base + 0x41.7018	8 bytes
UPA Reset Control Register	JBUS_Base + 0x41.7020	8 bytes
GPIO 0 Register	JBUS_Base + 0x46.0000	1 byte
GPIO 1 Register	JBUS_Base + 0x46.0001	1 byte
GPIO 2 Register	JBUS_Base + 0x46.2000	1 byte
GPIO 3 Register	JBUS_Base + 0x46.2001	1 byte
GPIO Data Register	JBUS_Base + 0x46.4000	8 bytes
GPIO Control Register	JBUS_Base + 0x46.4008	8 bytes
UPA Slot0 Configuration Register	JBUS_Base + 0x48.0000	8 bytes
UPA Slot1 Configuration Register	JBUS_Base + 0x48.0008	8 bytes
UPA Interface Configuration Register	JBUS_Base + 0x48.0010	8 bytes
PCI CSR Space		
PCI Performance Monitor Control Register	PCI_CSRBase+0x00.0100	8 bytes
PCI Performance Counter Register	PCI_CSRBase+0x00.0108	8 bytes
IOMMU Control Register	PCI_CSRBase + 0x00.0200	8 bytes
TSB Base Address Reg	PCI_CSRBase + 0x00.0208	8 bytes
IOMMU Flush Page Register	PCI_CSRBase + 0x00.0210	8 bytes
IOMMU Flush Context Register	PCI_CSRBase + 0x00.0218	8 bytes
Translation Fault Address Register	PCI_CSRBase + 0x00.0220	8 bytes
PCI Control/Status Register	PCI_CSRBase + 0x0.0000.2000	8 bytes
PCI PIO AFSR	PCI_CSRBase + 0x0.0000.2010	8 bytes

Register	Address	Access Size
PCI PIO AFAR	PCI_CSRBase + 0x0.0000.2018	8 bytes
PCI Diagnostic Register	PCI_CSRBase + 0x0.0000.2020	8 bytes
PCI Target Retry Limit	PCI_CSRBase + 0x0.0000.2030	8 bytes
PCI Target Latency Timer	PCI_CSRBase + 0x0.0000.2038	8 bytes
Scratch Pad Register	PCI-A_CSRBase + {0x0.0000.2040 - 0x0.0000.2238}	Any, provided within 8 byte boundary
Interrupt Routing Register	PCI-A_CSRBase + 0x0.0000.2240	8 bytes
I/O Cache Control/Status Register	PCI_CSRBase + 0x0.0000.2248	8 bytes
I/O Cache Tag Diagnostic Registers	PCI_CSRBase + {0x0.0000.2250 - 0x0000.2288}	8 bytes
I/O Cache Data Ram Diagnostic Registers	PCI_CSRBase + {0x0.0000.2290 - 0x0000.2488}	8 bytes
PCI Target Address Space Register	PCI_CSRBase + 0x0.0000.2490	8 bytes
PCI Target Error VA Log Register	PCI_CSRBase + 0x0.0000.2498	8 bytes
TLB Compare Setup Diag Reg	PCI_CSRBase + 0x00.A400	8 bytes
TLB Compare Result Diag Reg	PCI_CSRBase + 0x00.A408	8 bytes
TLB Tag Diag Regs	PCI_CSRBase + {0x00.A580 - 0x00.A5FF}	8 bytes
TLB Data RAM Diag Regs	PCI_CSRBase + {0x00.A600 - 0x00.A67F}	8 bytes
PCI Config CSR Space		
Vendor ID	PCI_ConfigBase + 0x00	2 bytes
Device ID	PCI_ConfigBase + 0x02	2 bytes
Command	PCI_ConfigBase + 0x04	2 bytes

Register	Address	Access Size
Status	PCI_ConfigBase + 0x06	2 bytes
Revision ID	PCI_ConfigBase + 0x08	1 byte
Programming I/F Code	PCI_ConfigBase + 0x09	1 byte
Sub-class Code	PCI_ConfigBase + 0x0A	1 byte
Base Class Code	PCI_ConfigBase + 0x0B	1 byte
Latency Timer	PCI_ConfigBase + 0x0D	1 byte
Header Type	PCI_ConfigBase + 0x0E	1 byte
Bus Number	PCI_ConfigBase + 0x40	1 byte
Subordinate Bus Number	PCI_ConfigBase + 0x41	1 byte
Extension Register	PCI_ConfigBase + 0x50	2 bytes
PCI Target Address Space Register	PCI_ConfigBase + 0x54	1 byte
Ichip CSR Space		
Interrupt Mapping Register for interrupt INO	Ichip_CSRBase+ 0x00.1000+INO*8 / PCI_CSRBase+ 0x00.1000+INO*8	8 bytes
UPA Port 0 Interrupt Mapping Register (register is also mapped normally via INO)	Ichip_CSRBase+ 0x00.6000 / PCI_CSRBase+ 0x00.6000	8 bytes
UPA Port 1 Interrupt Mapping Register (register is also mapped normally via INO)	Ichip_CSRBase+ 0x00.8000 / PCI_CSRBase+ 0x00.8000	8 bytes
Clear Interrupt Register for interrupt INO	Ichip_CSRBase+ 0x00.1400+INO*8 / PCI_CSRBase+ 0x00.1400+INO*8	8 bytes
PCI Int State Diag Register	Ichip_CSRBase + 0x00.A800 / PCI_CSRBase + 0x00.A800	8 bytes
OBIO and Internl Int State Diag Register	Ichip_CSRBase + 0x00.A808 / PCI_CSRBase+0x00.A808	8 bytes

Register	Address	Access Size
Interrupt Retry Register	Ichip_CSRBase + 0x00.1A00 / PCI_CSRBase+0x00.1A00	8 bytes
I-chip DMA Flush/Sync Complete Register	Ichip_CSRBase + 0x00.1A10	8 bytes
I-chip DMA Flush/Sync Pending Register	Ichip_CSRBase + 0x00.1A18	8 bytes

CHAPTER 5

Error Handling and Logging

JIO provides error detection, correction, and reporting capabilities. This chapter describes how JIO deals with errors.

5.1 Overview

This chapter describes the error detection, correction, and error reporting mechanisms supported by JIO.

Detected errors are classified as either fatal errors or non-fatal errors. Fatal errors may result in a system reset if enabled by software. Actions taken on non-fatal errors include: generating interrupts, setting status register bits, or none at all.

5.2 Error Detection and Reporting

5.2.1 *Error Detection*

The errors detected by JIO can be classified into the following :

i) *Jbus Errors*. These errors are logged in the Jbus Error Log Register(JbusCSRBase + 0x41.0018) . There are individual error reporting enables and global error report enable for these errors in Jbus Error Control Register(JbusCSRBase + 0x41.0008). There are individual interrupt enables and global error interrupt enable for these errors in Jbus Interrupt Control Register(JbusCSRBase + 0x41.0010).

ii) *PCI bus errors.*

- PCI bus errors detected during PIOs with JIO as a master can be due to TRDY timeout,excessive retries, master abort, target abort and bad data parity detection.
  - TRDY timeout and excessive retry errors get logged in PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000) bits [38:37] and in PCI PIO AFSR(PCI_CSRBase+0x0.0000.2010). The error address gets logged in PCI PIO AFAR(PCI_CSRBase+0x0.0000.2018). An interrupt gets issued if ERRINT_EN bit in PCI Control and Status Register is set.
  - Master abort ,target abort and data parity errors with JIO as the master get logged in both in PCI Configuration Status register (PCI_ConfigBase + 0x06) and in PCI PIO AFSR (PCI_CSRBase + 0x0.0000.2010).The error address gets logged in PCI PIO AFAR (PCI_CSRBase + 0x0.0000.2018). For master abort and target abort cases, an interrupt gets issued if ERRINT_EN bit in PCI Control and Status Register is set. For data parity error on a PIO write , an interrupt gets issued if ERRINT_EN bit in PCI Control and Status Register is set and if PER bit in the PCI Command Register is set.
- PCI bus errors detected during DMAs with JIO as a target can be due to address parity errors, data parity errors and PCI Retry Timeout.
  - Address parity errors on a DMA write or read with JIO as a target get logged in PCI Configuration Status Register(PCI_ConfigBase + 0x06). Also the error VA gets logged in PCI Target Error VA Log Register (PCI_CSRBase +0x0.0000.2498). An interrupt gets issued if ERRINT_EN bit in PCI Control and Status register is set.
  - Data parity error on a DMA write with JIO as a target gets logged in PCI Configuration Status Register(PCI_ConfigBase + 0x06).Also the error VA gets logged in PCI Target Error VA Log Register (PCI_CSRBase +0x0.0000.2498). An interrupt gets issued if ERRINT_EN bit in PCI Control and Status register is set and PER bit in the PCI Command Register is set.
  - Data parity error on a DMA read with JIO as a target does not get logged in JIO , nor does it assert any interrupts.
  - PCI Retry Timeout Error gets logged in PCI Control and Status Register (PCI_CSRBase + 0x0.0000.2000) bit 62. An interrupt gets issued if ERRINT_EN bit in PCI Control and Status register is set.
- PCI system error (SERR# asserted by any PCI device on the PCI bus). JIO samples SERR# assertion on the PCI bus by any device, and sets Bit 34 in PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000) to indicate occurrence of SERR. An interrupt gets issued if PER bit in the PCI Command Register is set, and if ERRINT_EN bit in PCI Control and Status register is set.

iii) *DRAM specific errors (UE/CE)* . UE/CE on read return data from Sparc IIIi Memory gets logged in JIO in UE/CE Asynchronous Fault Status Registers (JbusCSRBase + 0x41.0030/0x41.0040), and the error address gets logged in UE/CE

Asynchronous Fault Address Register(JbusCSRBase + 0x41.0038/0x41.0048).The UE/CE Interrupt enables are in Parity Control Register(JbusCSRBase + 0x41.0020) bits [62:61].

iv) *IOMMU Errors*. These errors are logged in IOMMU Control Register (PCI_CSRBase+0x00.0200) bits [28:24], and PCI Control and Status Register (PCI_CSRBase+0x0.0000.2000) bit 36. The IOMMU error gets qualified as a PCI Error if MMU_INT_EN bit in PCI Control and Status Register (PCI_CSRBase +0x0.0000.2000) is set, and raises a PCI error interrupt if ERRINT_EN bit in PCI Control and Status Register (PCI_CSRBase+0x0.0000.2000) is set.

v) *UPA64S error*. The UPA64S Slot Vacant error is detected any time a PIO read or write is directed toward a UPA64S port which was detected to be unoccupied at the end of system reset.There are no error detection enable and error interrupt enable for this error.

The following sections describe all the above-mentioned JIO errors in detail.

5.2.1.1 Detectable JBus Errors

Bad Jbus Command

This error is detected when JIO sees an invalid Jbus command.Jbus logs the error and reports the error if its reporting is enabled, and generates an interrupt if the interrupt enable for this particular error is on. JIO ignores the transaction and does not enqueue it in any of it's input queues.

Jbus Data Parity Error

This error is detected when the Jbus J_ADP[3:0] does not match the calculated parity of the J_AD[127:0] bus signals for an Address Cycle or any Data cycles for which JIO is the target. JIO detects address and data parity errors only if DATA_PRTY_EN bit of Parity Control Register(JbusCSRBase + 0x41.0020) is 1'b1.

After logging the error, JIO issues an interrupt if the interrupt enable for this particular error is on, and reports an error if its reporting is enabled.

If the parity error happens on a PIO to JIO, the address or data does get written to JIO's input PIO queues In case of data parity error, the bad data can get propagated to PCI /UPA leaf, or to any of JIO's internal CSRs .

For address parity error, even though the address gets enqueued in one of JIO's input queues, a system reset is caused to reset the entire system. Jbus Address parity error is considered to be a fatal error on Jbus.

For data parity error on a PIO write with JIO as a target, JIO causes a fatal error reset on Jbus only if the interrupt for the Jbus PIO write data parity error is disabled to protect the system from running with data corruption.

Note – Thus, software should keep the interrupt for this error disabled for normal system run at customer site , so that JIO would cause a fatal error reset everytime this parity error happens. (As we do not want system to run with data corruption).Cause of the fatal reset would be the error bit logged itself in bit 13 of the Jbus Error Log register. However if the error happens frequently causing frequent system (soft) reset , and the customer starts complaining, software should turn the interrupt on, so that next time the error happens, JIO would not assert fatal error reset; but software would now get the interrupt and could not only look at bit 13 set , but would also be able to diagnose further into the code that is causing bad PIO and get closer to identifying the problem.

When data parity error occurs on a cacheable DMA read return packet to JIO, the line is installed in I (Invalid) state in I/O Cache , and a target abort gets issued to the PCI master if the originating PCI transaction was a read. If the originating PCI transaction was a write, the write data for the transaction is simply thrown away by JIO.

When data parity error occurs on on a non-cacheable read return packet to JIO, JIO issues a target abort to the external PCI master.

Jbus Control Parity Error

This error is detected when the current value on J_PAR does not match the calculated parity of the Jbus Control signals driven on Jbus 2 Jbus clocks ago.

JIO detects control parity error only when “Control Parity Error Check Enable” bit (bit 43) of JIO Control/Status Register(JbusCSRBase + 0x41.0000) is 1’b1.

Upon detecting and logging the error, JIO signals fatal error on Jbus by asserting 3’b111 on its j_pack output for 4 consecutive system clocks. JIO with JPID[2:0] = 6 causes a system Soft-Reset when it detects fatal error asserted by any jbus port (including itself).

Jbus reports the error if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on.

Jbus Unmapped Error

For all cases of occurrences of unmapped error, Jbus logs the error first and reports it if reporting for unmapped error is enabled and generates an interrupt if the interrupt enable for unmapped error is set.

This error is detected by JIO as a target when the address for a PIO read or write is not in the range supported by JIO as per programmed values in Offset Mask and Base registers, or for a PIO NCWR / NCWRC transaction PA[42] != 1, or a CSR access is made to UPA CSRs while PAD_G_UPA_PCI_SEL = 0 or a CSR access is made to PCI-B CSRs while PAD_G_UPA_PCI_SEL = 1. For PIO writes , JIO discards the data and logs unmapped error. For PIO reads, JIO returns read error on Jbus with the error encoding indicating unmapped(J_ADTYPE bits [7:6] for read data return packet from JIO would indicate Read Error with J_AD[2:0] encoding indicating an Unmapped Error) .

Note – JIO ignores NCRD / NCBRD transaction with PA[42] != 1. So that if the target port is JIO or another CPU which does not exist in the system, the requesting port (CPU) would eventually time out and cause a fatal error reset. JIO can never generate a NCRD / NCBRD transaction with PA[42] != 1, so the only way such a transaction can get generated is from CPU through bad software or due to CPU hardware issues. However this is considered illegal address and should never happen in user and kernel code in production systems.

Note – Non-cached block write (NCBWR) with PA[42] != 1 causes incoming transaction decoder state machine in JIO to hang. This causes JIO to hang on subsequent transactions meant for JIO and also for transactions (e.g cacheable reads) meant for a different Jbus agent but to which JIO should still respond with a j_pack . This leads to system hang. This bug would not be fixed in JIO . This is because the proper fix to log the error and throw away the data is too difficult to do in ECO and would require resynthesis.So it is better to hang than let a bad write go unlogged / unreported.JIO can never generate a NCBWR with PA[42] != 1 . Though it is possible to have CPU issue NCBWR with PA[42] = 1'b0, it is considered illegal address and should never happen in user and kernel code in production systems.

Unmapped error is also detected by JIO when JIO receives a Jbus data packet for a DMA read with Read Error indicating an Unmapped Error.

When unmapped error is indicated on a cacheable DMA read return packet to JIO, the line is installed in I (Invalid) state in I / O Cache , and a target abort gets issued to the PCI master if the originating PCI transaction was a read. If the originating PCI transaction was a write, the write data for the transaction is simply thrown away by JIO.

When unmapped error is indicated on a non-cacheable read return packet to JIO, JIO issues a target abort to the external PCI master.

Unmapped error is also detected by JIO if JIO issues a write or read to an invalid Jbus port in the system. In that case, JIO dispatches the transaction as normal but logs unmapped error.

Jbus Timeout Error

This error is detected when J_ADTYPE bits [7:6] for an incoming read return data packet indicate Read Error and J_AD[2:0] encoding indicates a Timeout Error.

For all cases of occurrences of timeout error, Jbus logs the error first and reports the error if reporting for timeout error is enabled and generates an interrupt if the interrupt enable for unmapped error is set.

When timeout error is indicated on a IOMMU TTE data return packet to JIO,the IOMMU installs the entry in the IOMMU CAM and sets ERR = 1 and ERRSTS = Timeout. The error and error status bits in IOMMU Control Register also gets updated to reflect the occurrence of the Timeout Error. A target abort is issued for a DMA read, but for a DMA write, the write data for the transactions gets thrown away by JIO.

When timeout error is indicated on a cacheable DMA read return packet to JIO, the line is installed in I (Invalid) state in I/O Cache , and a target abort gets issued to the PCI master if the originating PCI transaction was a read. If the originating PCI transaction was a write, the write data for the transaction is simply thrown away by JIO.

When timeout error is indicated on a non-cacheable read return packet to JIO, JIO issues a target abort to the external PCI master.

Jbus Timeout Counter Expired Error

This error is detected when any of the two Jbus Timer Counters in PCI and GR sides expire on an outstanding read return. Jbus logs the error first and reports the error if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on.

If timeout counter expires on a IOMMU TTE read, he IOMMU installs the entry in the IOMMU CAM and sets ERR = 1 and ERRSTS = Timeout. The error and error status bits in IOMMU Control Register also gets updated to reflect the occurrence of the Timeout Error. A target abort is issued for a DMA read, but for a DMA write, the write data for the transactions gets thrown away by JIO.

If timeout counter expires on a cacheable DMA read return packet to JIO, the line is installed in I (Invalid) state in I/O Cache , and a target abort gets issued to the PCI master if the originating PCI transaction was a read. If the originating PCI transaction was a write, the write data for the transaction is simply thrown away by JIO.

If timeout counter expires on a non-cacheable read return packet to JIO, JIO issues a target abort to the external PCI master.

Jbus Bus Error

This error is detected when J_ADTYPE bits [7:6] for an incoming data packet indicate Read Error and J_AD[2:0] encoding indicates a Bus Error.Jbus logs the error first and reports it if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on.

When Bus error is indicated on a cacheable DMA read return packet to JIO, the line is installed in I (Invalid) state in I/O Cache , and a target abort gets issued to the PCI master if the originating PCI transaction was a read. If the originating PCI transaction was a write, the write data for the transaction is simply thrown away by JIO.

When Bus error is indicated on a non-cacheable read return packet to JIO, JIO issues a target abort to the external PCI master.

Jbus Illegal Byte Enable Error

This error is detected when the read byte mask is not correct for a NCRD with JIO as the destination.**1, 2, 4, 8,** and **16** bytes are expected to be read with NCRD, and the byte location is specified with a byte mask. The address is expected to be byte, half-word, word, dword, or 16-byte aligned. If JIO detects a violation of this rule, Jbus logs the error first and reports it if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on.

Note – Note that this is a requirement only for CPU initiated PIOs . For PCI to PCI DMA involving JIO, it is possible to violate this rule and cause an error. So to accomodate all masters, the overall recommendation is to turn this error reporting off during normal system operations.

Jbus Illegal Coherency Install State Error

This error is detected when an incoming read data return packet to JIO for a PCI DMA read or write has an install State of O which is illegal. Jbus logs the error first and reports it if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on.The line fill happens as regular in I/O Cache.

Snoop Error

This error is detected when the coherency state of a line in the I/O cache is wrong. Jbus logs the error first and reports it if its reporting is enabled and generates an interrupt if the interrupt enable for this particular error is on. Please refer to Jbus Spec for definition of Invalid/Erroneous Coherency states.

Jbus Fatal Errors

JIO with JPID[2:0] = 6 causes a system Soft-Reset by asserting J_RST_L when it detects an address parity error or control parity error or a data parity error on a PIO write to any of the two JIOs. These errors are treated as fatal errors as they cause system reset.

JIO would detect address and data parity errors only if DATA_PRTY_EN bit of Parity Control Register(JbusCSRBase + 0x41.0020) is 1'b1.

Note – For a detected address parity error , fatal error assertion happens irrespective of the state of the corresponding error reporting enable bit (bit 5 of Jbus Error Control Register [JbusCSRBase + 0x41.0008])

For a detected data parity error on a PIO write with JIO as a target, fatal error would get asserted on Jbus only when the interrupt for the error is disabled as follows :

- JE_INTEN bit (bit 63) of Jbus Interrupt Control register (JbusCSRBase + 0x41.0010) is 1'b0 (Global error interrupt enable for all Jbus specific errors is disabled)

or

- JE_INTEN bit (bit 63) of Jbus Interrupt Control register (JbusCSRBase + 0x41.0010) is 1'b1 but bit 13 of Jbus Interrupt Control register (JbusCSRBase + 0x41.0010) is 1'b0. (Global error interrupt enable for all Jbus specific errors is enabled but interrupt enable for the write data parity error is disabled)

Note – On detecting a data parity error on a PIO write, JIO would assert fatal error irrespective of the state of the corresponding error reporting enable bit (bit 13 of Jbus Error Control Register [JbusCSRBase + 0x41.0008])

JIO would detect control parity error and assert fatal error only when "Control Parity Error Check Enable" bit (bit 43) of JIO Control / Status Register(JbusCSRBase + 0x41.0000) is 1'b1.

Note – On detecting a control parity error, JIO would assert fatal error irrespective of the state of the corresponding error reporting enable bit (bit 12 of Jbus Error Control Register [JbusCSRBase + 0x41.0008])

Either JIO can assert fatal error on Jbus like any other Jbus port in the event of fatal error detected in the chip by asserting 3'b111 on its j_pack output for 4 consecutive system clocks.

But only JIO with JPID[2:0] = 6 causes a system Soft-Reset when it detects fatal error from any other Jbus port in the system by sampling 3'b111 on j_pack inputs for 4 consecutive system clocks.

5.2.1.2 *Detectable PCI Bus Errors*

PCI Receive Data Parity Error

This error is detected when a parity error is detected on either of the two sets of parity protected signals on the PCI bus (standard 32-bit data/control, and 64-bit data/control extensions), only during a valid data phase of a transaction (IRDY# and TRDY# asserted) involving JIO. Note that parity errors will not be detected during wait states (neither master nor target wait states). PCI receive data parity errors can be detected during the following transactions:

- PIO reads from a PCI device
- DMA writes by a PCI device

When JIO detects bad parity on incoming data on a PIO read, it sets DPE bit in the PCI Configuration Status register. It also sets DPAR and asserts PERR# only if PER bit in the PCI Command Register is set. JIO does not assert PCI error interrupt in this case but sends back Read Error = Bus Error on a JBus read return packet to the CPU that initiated the read if PER bit in the PCI Command Register is set. If PER = 0, the data with bad parity is returned to the CPU instead of Read Error.

When JIO detects bad parity on incoming data on a DMA write, it sets DPE bit in the PCI Configuration Status register and also asserts PERR# only if PER bit in the PCI Command Register is set.It also asserts error interrupt if PER bit in the PCI Command Register is set and ERRINT_EN bit in PCI Control and Status register is set.

Note – When PCI logic in JIO detects data parity error on DMA write with JIO as the target, it sends the bad data to the I/O Cache, and eventually the Sparc IIIi memory gets silently corrupted with the bad data. An error interrupt would get issued by JIO if ERRINT_EN bit in PCI Control and Status register is set, so that software would know about the occurrence of the data parity error. JIO logs the VA for the transaction which has data parity error in PCI Target Error VA Log Register (PCI_CSRBase+0x0.0000.2498). So the expectation is that software would abort any processes that were using the error address reported.

PCI Send Data Parity Error

This error is detected when the PCI PERR# signal is asserted by the other device during a PCI transaction involving JIO. This can be detected during the following transactions:

- PIO writes to a PCI device
- DMA reads by a PCI device

When JIO as a master (doing PIO writes to a PCI device) detects the PCI PERR# signal asserted by the other device while , it sets the DPAR bit in its PCI configuration status register , if the PER bit in the PCI Command Register is set .It also sets P_PERR or S_PERR bit in it's PCI PIO Asynchronous Fault Status Registerand logs the address in PCI PIO Asynchronous Fault Address register. It *does not* set DPE bit in the PCI Configuration Status register. If the PER bit is not set, JIO as a master ignores PERR# assertion of another device, and does not set DPAR . JIO also asserts error interrupt if ERRINT_EN bit in PCI Control and Status register is set, PERR# is asserted and DPAR is set.

If data parity error is detected by the external PCI device on a DMA read, and it asserts PERR#, JIO does not set DPE bit, nor does it set DPAR (because JIO is now acting as a target, and DPAR bit can get set only when JIO is a master). JIO does not also generate PCI error interrupt in this case. The assumption is that the external master after having detected the data parity error will assert a PCI interrupt and notify software of the error.

PCI Address Parity Error

When JIO detects PCI address parity error on bits [31:0] of 32/64 bit SAC or 32/64 bit DAC , it sets DPE bit in the PCI Configuration Status registerr.JIO asserts SERR# and sets SSE and STA bits in the PCI Configuration Status register if both SERR_EN and PER bits in the PCI Command Register are set. The setting of the STA bit causes a Target Abort to be issued by JIO. JIO also asserts error interrupt if SSE and STA bits get set and SERR# gets asserted , if ERRINT_EN bit in PCI Control and Status register is set.

Note – When PCI logic in JIO detects address parity error on DMA write with JIO as the target, it throws away all the data for that particular DMA write transaction. Thus JIO I/O Cache does not receive any bad data from JIO PCI logic on address parity error , thereby avoiding eventual corruption of data in Sparc IIIi memory.

JIO does not detect address parity errors on bits[63:32] of SAC 32/64 bit or 32/64 bit DAC , and hence does not set any CSR bits, nor does it assert SERR#.

PCI System Error (SERR#)

This error is detected any time that the PCI SERR# signal is asserted by any PCI device. Possible reasons for a device to assert SERR# include: detection of an address parity errors, or device specific fatal errors. When JIO detects PCI SERR# signal assertion on the PCI bus by any PCI device, it sets the PCI_SERR bit in the PCI Control and Status Register. JIO asserts PCI error interrupt if PER bit in the PCI Command Register is set and ERRINT_EN bit in PCI Control and Status register is set.

PCI Target-Abort

This error is detected when, during a PCI transaction for which JIO is the master, the target device asserts STOP# and deasserts DEVSEL#. Reasons that a target might issue a target-abort include: unsupported byte enables, an unsupported addressing mode, an address parity error, and device specific errors. A target-abort may be detected during any PIO reads or writes to PCI devices.

PCI Master-Abort

This error is detected when JIO begins a PCI transaction, and no device responds by asserting DEVSEL# within X cycles. This is typically because no device is mapped at the specified address. A Master-Abort error can be detected during any PIO read or write attempt to a PCI device.

PCI TRDY# Timeout error

When JIO is a PCI master doing a PIO read or write, and the target device terminates the transaction with a retry more than the specified number of times in a row, JIO will abort the transaction and signal this error. If this error is detected on a PIO write , JIO asserts error interrupt if ERRINT_EN bit in PCI Control and Status register is set.

PCI Retry Limit Error

This error is detected when JIO initiates a PCI transaction, and the PCI target device terminates the transaction with a Retry (disconnects with no data transfer) for XXXX successive attempts. This error can be detected during any PIO read or write to a PCI device. If this error is detected on a PIO write , JIO asserts error interrupt if ERRINT_EN bit in PCI Control and Status register is set.

PCI Retry Timeout (PCI_DTO_ERR)

When JIO is the target of a PCI read transaction, and JIO turns it into a Delayed Read by issuing retry on PCI (reserves resources, and issues the request to Jbus), and the PCI master does not reissue the same transaction within 2^15 clocks after data has been returned to the PCI leaf.

Note: In case of 2 masters on a PCI bus doing DMA write and read and DMA read respectively to the same address with JIO as the target, and the master doing the DMA read gets in first but misses in I/O cache and issues a retry, it is not possible for JIO to guarantee correct data being returned to the second master doing a Write and Read to the same address, as there is no mechanism for tagging any read with the requesting master. The second master's read may get the cached data for the first read if it shows up prior to the delayed read for the first read, since they are to the same address.

5.2.1.3 Detectable ECC Errors

Jbus ECC Error

JIO does not detect ECC. Only memory controllers detect ECC. However, UE and CE Errors are reported by J_ADTYPE bits [6:3]. If JIO detects these errors in a read data return data packet, Jbus cluster of JIO logs the error first and then generates an interrupt if the interrupt enable for this particular error is on.

If data is returned from Sparc IIIi memory with UE in it, Jbus cluster in JIO still sends the data to I/O Cache and I/O Cache installs the data and updates the tag to I (Invalid) state. If the PCI transaction that caused the Jbus read was a read, JIO asserts a target abort on detecting the UE.If the PCI transaction was a write, JIO throws away all the data from that PCI DMA write transaction.Thus the external PCI device is not allowed to read or write a line that is certified to be bad (UE) in Sparc IIIi memory.

If data is returned from Sparc IIIi memory with CE in it, it indicates the data had a Correctable Error which has been corrected by hardware. On receiving data with CE marked on it, Jbus cluster in JIO sends the data to I/O Cache and I/O Cache installs the data and updates the tag in the correct state. Thus the fill happens like a normal fill. However from that time onwards till the interrupt is issued and software services the interrupt, the external PCI master is allowed to read and write the line that had CE in it as normal. Software will probe the location (by looking at the PA logged in the CE AFAR in JIO) to see if the error is persistent.If it is , it will correct it using Sparc IIIi , to avoid getting multiple interrupts for one CE.

5.2.1.4 *Detectable IOMMU Errors*

IOMMU Translation Error

This error can be detected during the address translation of any DMA read or write transaction. It is detected if any of the following occur:

- The PCI virtual DMA address is outside the range created by aligning the TSB-defined space to the top of the 512 Mbyte chunks of enabled Target Address Space (only for TSB and TBW defined VA spaces smaller than 512 Mbytes).
- Illegal combination of TSB_SIZE and TBW_SIZE
- The Valid bit is not set in the TTE for the given virtual page
- The DMA operation is a write, and the TTE is marked read-only
- Timeout Error on TTE read
- UE on TTE read

5.2.1.5 *Detectable UPA64S Errors*

UPA64S Slot Vacant

This error is detected any time that a PIO read or write is directed toward a UPA64S port which was detected to be unoccupied at the end of system reset.

P_REPLY has a pull-up which pulls it high for an unoccupied UPA64S port. At reset deassertion, the status of the P_REPLY (reflected in “Slot Empty” bit of the UPA slot configuration register) is sampled. If this bit is a 1, any subsequent attempt to issue a PIO read or Write to this Slot is treated as an Error. A write fails like a NOP, for a read, “read error” is sent back, error type being Timeout.

5.2.2 Error Reporting

5.2.2.1 Summary of Error Reporting

TABLE 5-1 summarizes the response of JIO when an error is detected. All JIO operations involve the Jbus bus and one other leaf bus or leaf block. Errors typically occur and are detected on a single bus, but error responses may occur on neither, one, or both of the involved buses. Only the immediate response of JIO is shown

below, and not that of any other devices that may be involved. For example, when a Jbus Data ECC Error is detected during a DMA Read from a PCI device, the table shows that JIO issues a Target-abort, but does not indicate that the PCI device then typically issues an interrupt as a result. In all cases where JIO responds with an interrupt, the interrupt is only issued if enabled.

The following register abbreviations are used in the table. See Chapter 4 “Configuration and Status Registers for further details:

- JIB CSR - Jbus Error Log Register(JbusCSRBase + 0x41.0018)
- JIB INTEN CSR - Jbus Interrupt Control register (JbusCSRBase + 0x41.0010)
- JIB CE/UE AFSR - UE/CE Asynchronous Fault Status Registers(JbusCSRBase + 0x41.0030/0x41.0040)
- JIB CE/UE AFAR - UE/CE Asynchronous Fault Address Register(JbusCSRBase + 0x41.0038/0x41.0048)
- PCI Status - PCI Config Status Register(PCI_ConfigBase + 0x06)
- PCI CSR - PCI Control and Status Register(PCI_CSRBase+0x0.0000.2000)
- PCI AFAR - PCI PIO Asynchronous Fault Address Register (PCI_CSRBase+0x0.0000.2010)
- PCI AFSR - PCI PIO Asynchronous Fault Status Register (PCI_CSRBase+0x0.0000.2018)
- PCI VA Log - PCI Target Error VA Log Register(PCI_CSRBase+0x0.0000.2498)
- MMU TFAR - Translation Fault Address Register (PCI_CSRBase+0x00.0220)
- MMU CSR - IOMMU Control Register(PCI_CSRBase+0x00.0200)
- UPA CSR - UPA64S Configuration Register(JbusCSRBase + 0x48.0000/0x48.0008)

TABLE 5-1 Summary of Error Reporting

Src/Dst Bus/Leaf	Transaction Type	Error Type	Jbus Response	Leaf Bus Response	Error Status Register[bit(s)]
Jbus	PIO Reads, PIO Writes	Address Parity Error	Jbus Error Interrupt System reset.	None	JIB CSR[5]
Jbus	PIO Writes	Jbus Data Parity Error	(1)System reset if JIB INTEN CSR [63,13] = 2’b10, or JIB INTEN CSR[63] = 1’b0. (2) Jbus Error Interrupt if JIB INTEN CSR [63,13] = 2’b11.	None	JIB CSR[13]
PCI	DMA Read	Jbus Data Parity Error	Jbus Error Interrupt	Target-abort	JIB CSR[6] PCI Status[11] PCI VA Log
PCI	DMA Write	Jbus Data Parity Error	Jbus Error Interrupt	None	JIB CSR[6]

TABLE 5-1 Summary of Error Reporting

Src/Dst Bus/Leaf	Transaction Type	Error Type	Jbus Response	Leaf Bus Response	Error Status Register[bit(s)]
N/A	Not Applicable	Jbus Control Parity Error	Jbus Error Interrupt, System Reset.	None	JIB CSR[12]
Jbus	PIO Read (NCRD)	Jbus Illegal Byte Enable Error	Jbus Error Interrupt	None	JIB CSR[10]
PCI	DMA Read	Jbus Illegal Coherency Install State Error	Jbus Error Interrupt	None	JIB CSR[8]
Jbus	RDD, RDS, RDSA, OWN, WRI, WRIS	Snoop Errors	Jbus Error Interrupt	None	JIB CSR[11,14,15, 16,17,19,20,21]
Jbus	Unknown	Bad Jbus Command	Jbus Error Interrupt	None	JIB CSR[62]
PCI	DMA Read	Jbus Unmapped Error	Transaction aborted. Jbus Error Interrupt	Target-abort	JIB CSR[4] PCI Status[11] PCI VA Log
PCI	DMA Write	Jbus Unmapped Error	Jbus Error Interrupt	None	JIB CSR[4]
PCI	DMA Read	Jbus Timeout Error	Transaction aborted. Jbus Error Interrupt	Target-abort	JIB CSR[1,3] PCI Status[11] PCI VA Log
PCI	DMA Write	Jbus Timeout Error	Jbus Error Interrupt	None	JIB CSR[1,3]
PCI	DMA Read	Jbus Bus Error	Transaction aborted. Jbus Error Interrupt	Target-abort	JIB CSR[2] PCI Status[11] PCI VA Log
PCI	DMA Write	Jbus Bus Error	Jbus Error Interrupt	None	JIB CSR[2]
PCI	DMA Read	Jbus Data ECC Correctable Error	CE Interrupt	None	JIB CE AFSR [62,59] JIB CE AFAR [42:4]
PCI	DMA Write	Jbus Data ECC Correctable Error	CE Interrupt	None	JIB CE AFSR [62,59] JIB CE AFAR [42:4]

TABLE 5-1 Summary of Error Reporting

Src/Dst Bus/Leaf	Transaction Type	Error Type	Jbus Response	Leaf Bus Response	Error Status Register[bit(s)]
PCI	DMA Read	Jbus Data ECC Uncorrectable Error	UE Interrupt	Target-abort	JIB UE AFSR [62,59] JIB UE AFAR [42:4]
PCI	DMA Write	Jbus Data ECC Uncorrectable Error	UE Interrupt	None	JIB UE AFSR [62,59] JIB UE AFAR [42:4]
PCI	None	PCI System Error	PCI Error Interrupt	None	PCI CSR[34]
PCI	Unknown (DMA)	PCI Address Parity Error	PCI Error Interrupt	Target-abort	PCI Status [14,15] , PCI VA log
PCI	PIO Read	PCI Master-abort	Read Error =Timeout	None	PCI Status[13]
PCI	PIO Write	PCI Master-abort	PCI Error Interrupt	None	PCI Status[13] PCI AFSR[63,57], PCI AFAR[31:0]
PCI	PIO Write (PCI Special Cycle)	PCI Master-abort (This is the normal termination for PCI Special Cycles)	None	None	None
PCI	PIO Read	PCI Retry Limit Error	Read Error =Timeout	Stop retrying	PCI CSR[37]
PCI	PIO Write	PCI Retry Limit Error	PCI Error Interrupt	Stop retrying	PCI CSR[37] PCI AFSR[61,55], PCI AFAR[31:0]
PCI	PIO Read	PCI TRDY# Timeout error	Read Error = Timeout	Abort transaction	PCI CSR[38]
PCI	PIO Write	PCI TRDY# Timeout error	PCI Error Interrupt	Abort transaction	PCI CSR[38] PCI AFSR[59,53] PCI AFAR[31:0]
PCI	PIO Read	PCI Target-abort	Read Error =Bus Error	None	PCI Status[12]
PCI	PIO Write	PCI Target-abort	PCI Error Interrupt	None	PCI Status[12], PCI AFSR[62,56], PCI AFAR[31:0]
PCI	DMA Read	PCI Retry Timeout	PCI Error Interrupt	Free delayed transaction resources	PCI CSR[62]
PCI	DMA Write	PCI Receive Data Parity Error	PCI Error Interrupt	Assert PERR#	PCI Status[15], PCI VA Log

TABLE 5-1 Summary of Error Reporting

Src/Dst Bus/Leaf	Transaction Type	Error Type	Jbus Response	Leaf Bus Response	Error Status Register[bit(s)]
PCI	PIO Write	PCI Send Data Parity error	PCI Error Interrupt	None	PCI Status[8], PCI AFSR[60,54], PCI AFAR[31:0]
PCI	PIO Read	PCI Receive Data Parity Error	Read Error =Bus Error	Assert PERR#	PCI Status[15]
PCI	DMA Read	PCI Send Data Parity error	None	None	PCI Status[8]
PCI	DMA Read	IOMMU Translation Error	IOMMU Error Interrupt	Target-abort	PCI Status[11] PCI CSR[36] MMU CSR[28:24] PCI VA Log(VA) MMU TFAR(PA) (for invalid error only)
PCI	DMA Write	IOMMU Translation Error	IOMMU Error Interrupt	None	PCI CSR[36] MMU CSR[28:24] MMU TFAR(PA) (for Invalid and Protectionerrors)
UPA64S	PIO Read	UPA64S Slot Vacant Error	Ignore transaction	Read Error =Timeout	UPA CSR[63]
UPA64S	PIO Write	UPA64S Slot Vacant Error	Ignore transaction	None	UPA CSR[63]

5.3 Undetected Errors

Certain error conditions are not detected or reported by the JIO. Examples of these errors are listed below.

Note – This list may not enumerate all unreported errors.

- A PIO write to a read-only register is ignored.
- A read from a write-only register returns unknown data.
- Certain PCI protocol violations (device responding with DEVSEL#, but never responding with TRDY# or STOP#).
- Infinite NACKing of an interrupt

5.4 Issues

- Should various Jbus errors cause a Jbus error interrupt (otherwise they only cause status bit changes, and may not propagate to OS)? Yes
- What to do on address parity errors (both Jbus and PCI) - do we detect in time to ignore the transactions? Yes.
- Do PIO writes errors need to get reported? Yes
- UPA64S PIO Write to vacant slot - need to signal / drain data from switch? For JIO UPA data will come with address, so not an issue.

CHAPTER 6

Performance

This chapter talks about JIO's Peripheral Component Interconnect (PCI) DMA and PIO memory performance.

6.1 Overview

This chapter describes the PBM and I/O Cache prefetch mechanisms in some detail and the role they play in optimizing JIO's DMA read performance.

This chapter also enumerates JIO's DMA performance for 66/33 MHz PCI bus frequencies, 64/32 bit bus masters for different burst sizes with one and two masters per bus.

This chapter also gives some recommended settings in PCI and I/O cache CSRs to get the most optimal DMA performance for 1 and 2 master cases that are tabulated.

Also discussed are the architectural limitations to optimal DMA performance for the 2 and 3 master case for 66 MHz 64 bit bus cases, and the performance sustained by JIO for the less stringent 33 MHz 32 bit case for 3 masters.

Finally the chapter describes JIO's sustained memory PIO performance on each PCI bus for 66/33 MHz PCI bus operation and 64/32 bit client.

6.2 PBM and I/O Cache Prefetch

JIO supports a two-level prefetch mechanism (in PBM and I/O Cache modules respectively) to optimize DMA read performance under the assumption that most DMA traffic from external PCI bus master would be sequential in nature.These two prefetch mechanisms work hand in hand to optimize the DMA read throughput with JIO as the target.

6.2.1 PBM prefetch

JIO's PBM has a two line deep read data storage buffer that gets filled by data coming from the I/O Cache on reads.

PBM requests data to the I/O Cache, gets the data and puts it in the buffer. The I/O Cache has the ability to entertain request for 2 lines from the PBM at any point of time. So long as PBM has the buffer space to get more than one line of data from the I/O Cache, and the PBM prefetch is enabled, it sends a new request to I/O Cache upon the receipt of the acknowledgement for the prior request from I/O Cache.

So if there is nothing in the PBM buffer, and it sends a new request to the I/O Cache, the I/O Cache accepts the request and returns data to PBM (if it has the line) or sends a request to Jbus to get the data from memory.If the PBM prefetch is enabled, the PBM is going to ask for the second line and keep installing the seond line in its buffer, while the first line is being drained from its buffer to the PCI bus. So that when the next line is requested from the PCI bus, the data will get drained directly from the buffer and would save a round - trip delay to the I/O Cache and back. And the I/O Cache is going to ask the PBM to pick up both the current line and the next line since the PBM has asked for it.

So the idea is to get the next line to the buffer in advance if there is space in the PBM buffer to hide the latency to and from the I/O Cache to the PCI bus, and thus optimize the DMA read performance.

Thus PBM prefetch is always one line and always the next line if there is space in the PBM buffer. Like I/O Cache prefetch , the PBM prefetch can be selectively turned on/off based on the specific PCI read command from the PCI bus (read,read line, read multiple). Please refer to PCI Control and Status Register (PCI_CSRBase +0x0.0000.2000) for the different prefetch enable bits in the PBM for the different PCI commands.

Note – For DMA SAC accesses from the PCI master with JIO as the target , PBM detects 8K byte boundary crossing of the next PCI Virtual Address , by checking va[12:6] == 7'b1111111, and stops prefetching (does not request for the next line to I/O cache). However for DAC accesses, it does prefetch across 8K boundary.

6.2.2 I/O Cache Prefetch

JIO I/O Cache has similar ability to prefetch a maximum of 3 cache lines from the memory with the access that hit/missed in the I/O cache from the PBM. When it gets the requested line from Jbus, it installs it in the data RAM and also updates the tag RAM. To maximize performance, it also initiates data transfer to the PBM buffer as soon as it gets the first data packet from Jbus.

Xfer from JBU to IOC |=|=|=|=|

Xfer from IOC to PBM |=|=|=|=|

this is shown for a typical 4 cycle data return over Jbus (for 1 line)

When the I/O Cache gets the next line(s) due to the prefetch, it installs them , and keeps sending them over to the PBM in a similar way if the PBM buffer still has space and the PBM wants more lines (PBM prefetch enabled).

Like the PBM prefetch, the I/O Cache prefetch can be selectively turned on/off based on the specific PCI read command from the PCI bus (read,read line, read multiple). Please refer to I/O Cache Control and Status Register (PCI_CSRBase +0x0.0000.2248) for the different prefetch enable bits in the I/O Cache for the different PCI commands.

For I/O Cache, the number of lines that can be prefetched ranges from 1 to 3 and is programmable in the I/O Cache Control/Status Register for each individual PCI read command. Though due to Errata 15 in Sparc IIIi, the current recommendation is to restrict this to 1 line only. Moreover the performance with multiple masters on the PCI bus is reasonably good with one line of prefetch itself. Also the prefetch distance from the current line (prefetch stride) is programmable in the I/O Cache Control/Status Register. In the subsequent sections of this chapter, there are values recommended for this parameter as well to get good DMA performance.

Besides the read prefetch, JIO I/O Cache supports prefetch capability for read-modify-write operations as well. This prefetch if enabled in the I/O Cache Control/Status Register improves DMA write performance for the case of back to back strings of partial line writes with JIO as the target.For a long continuous DMA burst that spans several cache lines, this feature is not particularly useful, as JIO PBM and I/O Cache break the burst into 64 byte (cache line) chunks and dispatch writes to memory over Jbus (as WRIS Jbus commands) bypassing the I/O Cache altogether.

Note – For both read and partial line write from PBM, I/O Cache issues the prefetch (if enabled in I/O Cache control register) both on hit and miss of the read/write request from PBM, only if the calculated prefetch target PA does not already reside in the I/O Cache and the calculated prefetch target PA does not cross 8K byte boundary.

6.3 DMA Performance

6.3.1 1 Master Case

6.3.1.1 Recommended CSR Bit Settings

The recommended CSR settings to get optimal DMA performance for 1 master case are as follows:

- TRGT_RW_STL_WT field in PCI control and Status register to be set to 3'h7
- POFFSET field in I/O cache Control/Status Register to be set to 7'h01 (this will set the I/O cache prefetch stride to a value of 2).
- PLEN_RD_MLTPL, PLEN_RD_ONE,PLEN_RD_LINE fields in I/O cache control/status register should be each set to 0x00 (I/O cache prefetch length of 1 cache line for each of Read Multiple, Read and Read Line PCI commands).
- For the PCI driver, the recommendation is to use Read or Read Line PCI commands for less than or equal to 128 byte read bursts and Read Multiple PCI command for greater than 128 byte read bursts. PCI API prefetch should be turned on for Read Multiplies only, should be turned off for Read and Read Line commands. Thus the recommended CSR setting would be PEN_RD_MLTPL field in PCI Control and Status Register to 1'b1 and PEN_RD_ONE,PEN_RD_LINE fields each to 1'b0.
- WRT_PEN field in I/O cache Control/Status Register to be set to 1'b1 (enabling Prefetch for Partial Line Writes).

6.3.1.2 1 Master Performance Numbers

Simulation results for 1 master DMA performance are tabulated below in Megabytes/sec. The simulation conditions are as follows:

- The numbers were obtained using one PCI device attached to each leaf
- DMA reads and writes were done back to back in 9 block sizes (8, 16, 32, 64, 128, 512, 1024, 4096, 8192 bytes)
- Each of these tests was conducted on both leaves (A:66MHz, B:33MHz) with both a 32 bit and a 64 bit bus width
- Each of these tests was conducted as a single leaf and a single bus width at a time for 4 possible configurations
  - 66 MHZ 64 Bit
  - 66 MHz 32 Bit
  - 33 MHZ 64 Bit
  - 33 MHz 32 Bit
- During each of these tests there was no other traffic on the bus
- JBus frequency: 200 MHz, CPU frequency: 1.6 GHz, DRAM frequency: 133 MHz
- The memory latency for the Sparc IIIi system with 1 CPU and 1 JIO in the system was 119 nsec.

TABLE 6-1 1 Master 66 MHz PCI Bus width 64 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	105	52
16	211	96
32	301	162
64	384	248
128	444	281
512	452	370
1024	515	430
4096	515	430
8192	515	430

TABLE 6-2 1 Master 66 MHz PCI Bus width 32 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	105.5	48
16	150.5	81
32	192	124
64	222	168.5
128	241	183.5
512	258	235
1024	260	255
4096	260	260
8192	261	260

TABLE 6-3 1 Master 33 MHz PCI Bus width 64 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	52	37
16	105	66
32	150	105
64	192	150
128	222	168
512	239	231
1024	257	252
4096	258	259
8192	260	260

TABLE 6-4 1 Master 33 MHz PCI Bus width 32bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	52.5	33
16	75	52.5
32	96	75
64	111	96
128	120.5	103

TABLE 6-4 1 Master 33 MHz PCI Bus width 32bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
512	125	124
1024	130	128.5
4096	131	131
8192	131	131

Note – Although the numbers shown above are for 1 PCI bus only, simulation results show that for 1 master on each bus, JIO can sustain the same throughput on each bus if both buses are active simultaneously. Thus for 1 master connected to each PCI bus, JIO can sustain a peak write and read throughput of 1 gbyte/sec for the 66mHz/64 bit case for long bursts (>= 4 K bytes), with 500 MB/sec on each bus.

6.3.2 2 Master Case

6.3.2.1 Recommended CSR Bit Settings

The recommended CSR settings to get optimal DMA performance for 2 master case are as follows:

- TRGT_RW_STL_WT field in PCI Control and Status Register to be set to 3'h7
- POFFSET field in I/O cache Control/Status Register to be set to 7'h01 (this will set the I/O cache prefetch stride to a value of 2).
- PLEN_RD_MLTPL,PLEN_RD_ONE,PLEN_RD_LINE fields in I/O cache control/status register should be each set to 0x00 (I/O cache prefetch length of 1 cache line for each of Read Multiple, Read and Read Line PCI commands).
- For the PCI driver, the recommendation is to use Read or Read Line PCI commands for less than or equal to 128 byte read bursts and Read Multiple PCI command for greater than 128 byte read bursts. PCI API prefetch should be turned on for Read Multiplies only, should be turned off for Read and Read Line commands. Thus the recommended CSR setting would be PEN_RD_MLTPL field in PCI Control and Status Register to 1'b1 and PEN_RD_ONE,PEN_RD_LINE fields each to 1'b0.
- WRT_PEN field in I/O cache Control and Status Register to be set to 1'b1 (enabling Prefetch for Partial Line Writes).

6.3.2.2 2 Master Performance Numbers

Simulation results for 2 master DMA performance are tabulated below in Megabytes / sec. The simulation conditions are as follows:

- The numbers were obtained using two PCI devices attached to each leaf
- DMA reads and writes were done back to back in 9 block sizes (8, 16, 32, 64, 128, 512, 1024, 4096, 8192 bytes)
- Each of these tests was conducted on both leaves (A:66MHz, B:33MHz) with both a 32 bit and a 64 bit bus width
- Each of these tests was conducted as a single leaf and a single bus width at a time for 4 possible configurations
  - 66 MHZ 64 Bit
  - 66 MHz 32 Bit
  - 33 MHZ 64 Bit
  - 33 MHz 32 Bit
- During each of these tests there was no other traffic on the bus
- JBus frequency: 200 MHz, CPU frequency: 1.6 GHz, DRAM frequency: 133 MHz
- The memory latency for the Sparc IIIi system with 1 CPU and 1 JIO in the system was 119 nsec.

TABLE 6-5 2 Master 66 MHz PCI Bus width 64 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	105	48
16	211	88
32	301	150
64	384	230
128	444	272
512	504	200
1024	515	230
4096	519	230
8192	516	230

TABLE 6-6 2 Master 66 MHz PCI Bus width 32 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	105.5	44
16	150.5	75
32	192	117
64	222	162
128	241	179.5
512	257.5	208.5
1024	260.5	231.5
4096	260.5	255
8192	260.5	255

TABLE 6-7 2 master 33 MHz PCI Bus width 64 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	52	33
16	105	58
32	150	96
64	192	140
128	222	162
512	252	212
1024	257	229
4096	260	252
8192	259	256

TABLE 6-8 2 Master 33 MHz PCI Bus width 32bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	52.5	29
16	75	48
32	96	65
64	111	91.5
128	120.5	100.5

TABLE 6-8 2 Master 33 MHz PCI Bus width 32bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
512	128.5	118
1024	130	124
4096	130	128
8192	130	128

Note – There is some degradation for > 128 byte Read bursts with 2 masters at 64-bit/66 MHz. This is due to prefetch interaction issues between the two address streams in the I/O cache and the relatively fast data consumption rate for 64-bit/66 MHz.The prefetch cannot keep up with that rate when two DMA address streams ar e involved, and retries and disconnects are issued.

Note – The following is recommended: use 128 byte burst sizes for the 2 master, 64-bit 66 MHz case to get sustained good read performance without any retries/ disconnects.

6.3.3 3 Master Case

Simulation results for the 3 master case in the same simulation environment as the 1 and 2 master cases with the same settings in PCI and I/O cache control and status registers show significant degradation at 66 MHz/64 bit. This is due to a capacity issue in the I/O cache. To sustain the DMA bus throughput comfortably, the I/O cache needs 4 entrees per master. The I/O cache is fully set associative with 8 entrees. So it can accommodate 2 masters reasonably well, but with 3 masters there is a big performance degradation. Future JIO derivatives should look at expanding the I/O cache to 16 entrees to handle 3 and 4 masters / leaf.

However simulation for the 3 master case at 33 MHz/32 bit shows considerably better results, due to relatively much slower data consumption rate than the 66 MHz/64 bit case. Those results are tabulated below:

TABLE 6-9 3 master 33 MHz PCI Bus width 32 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
8	52.5	29
16	75	48
32	96	57.5

TABLE 6-9 3 master 33 MHz PCI Bus width 32 bits

Burst size (in bytes)	DMA write (MB/sec)	DMA Read (MB/sec)
64	111	66.5
128	120.5	69
512	128.5	117.5
1024	130	125
4096	130	128
8192	130	128

6.4 Memory PIO Performance

6.4.1 Memory PIO Bandwidth

Simulation results for PCI memory PIO performance are tabulated below in Megabytes/sec. The simulation conditions are as follows:

- The numbers were obtained using one PCI device attached to each leaf.
- PIO reads and writes were done in four block sizes (4, 8, 16, 64 bytes) for 32 bit client, and in four blocks sizes (4, 8, 16, 64 bytes) for 64 bit client.
- Each of these tests was conducted on both leaves with both a 32 bit and a 64 bit bus width
- Each of these tests was conducted as a single leaf and a single bus width at a time for 4 possible configurations
  - 66 MHZ 64 Bit
  - 66 MHz 32 Bit
  - 33 MHZ 64 Bit
  - 33 MHz 32 Bit
- During each of these tests there was no other traffic on the bus
- ARB_PARK bit in JIO Control and Status register [bit 16] programmed to 1’b1 (PCI bus grant stays asserted to the last requestor while the PCI bus is idle so that device is the default driver)

TABLE 6-10 JIO as master: mem PIO, 66 MHz PCI Bus width 64 bits

Burst size (in bytes)	PIO write (MB/sec)	PIO Read (MB/sec)
4	88	62
8	105.5	105.5
16	211	211
64	384	384

TABLE 6-11 JIO as master: mem PIO 66 MHz PCI Bus width 32 bits

Burst size (in bytes)	PIO write (MB/sec)	PIO Read (MB/sec)
4	88	62
8	105.5	105.5
16	150	150
64	222	222

TABLE 6-12 JIO as master: mem PIO, 33 MHz PCI Bus width 64 bits

Burst size (in bytes)	PIO write (MB/sec)	PIO Read (MB/sec)
4	44	33
8	52.5	52.5
16	105	105
64	192	192

TABLE 6-13 JIO as master: mem PIO, 33 MHz PCI Bus width 32bits

Burst size (in bytes)	PIO write (MB/sec)	PIO Read (MB/sec)
4	44	33
8	52.5	52.5
16	75	75
64	111	111

6.4.2 PIO Latency in JIO

The following table shows pin to pin PIO latency in JIO for NCRD and NCWR with Jbus running at 200 Mhz, and PCI bus as 66 Mhz .

TABLE 6-14 PIO Latency in JIO, Jbus at 200 Mhz, PCI bus at 66 Mhz

Jbus Command Type	Latency (nsec)
NCWR	Jbus to PCI bus (within JIO) : 83 nsec
NCRD	Jbus to PCI bus(within JIO) : 68 nsec PCI bus to Jbus(within JIO) : 136 nsec Total(within JIO) = 204 nsec

Note – These numbers do not take into account any data return latency overhead and PCI bus arbitration / protocol related overhead on the part of the PCI slave for returning data to JIO .These are strictly pin to pin latency numbers in JIO.

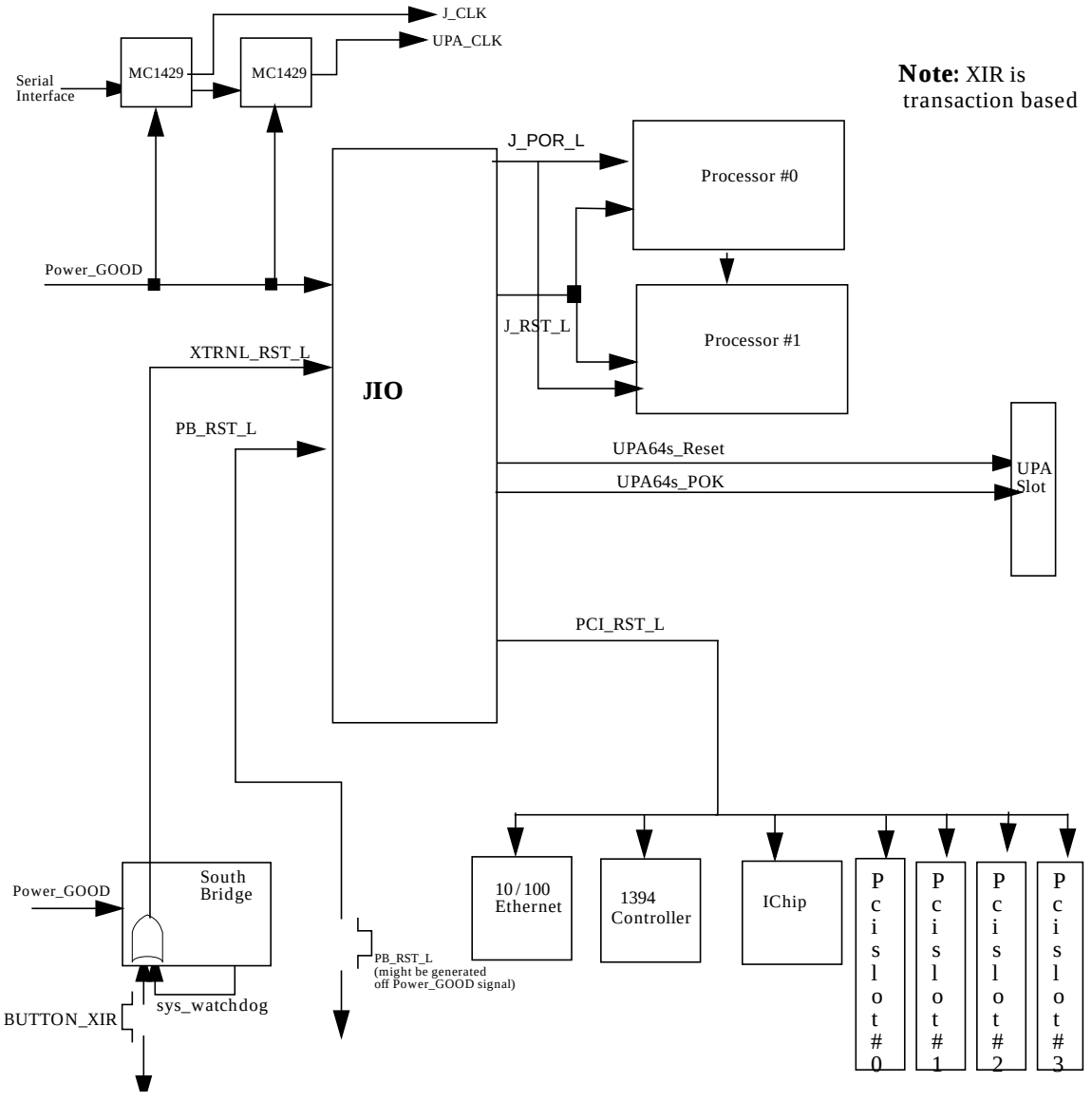
CHAPTER 7

Reset

7.1 Overview

The following shows a block diagram of reset throughout the system.

FIGURE 7-1 Reset Block Diagram



Sparc IIIi supports the SPARC V9 Power-On Reset (POR) trap. A processor takes this trap when its J_POR_L or J_RST_L pin is activated by an external event, because of signals received by JIO with JPID[2:0] = 6, on some of its inputs or because bits are set in some of its internal registers. Although there is a single POR trap in Sparc IIIi, there are two different variations of this trap.

7.1.1 Power-On Reset (Hard Reset)

When power is applied, JIOs detect the rising edge of the Power-Good (POK) signal from the power supply. JIO with JPID[2:0] = 6, resets the entire system and generates J_POR_L and J_RST_L to the processors and the other JIO. Sparc IIIi samples J_POR_L and takes (in Sparc IIIi's terminology) a "hard reset". Please refer to FIGURE 7-2 on page 186 for Power-On Reset.

There are two variants of the hard reset: **long** and **short**.

In order to create short and long "hard" reset for the Sparc IIIis, XTRNL_RST_L pin is toggled appropriately along with external POK. During POK 0 to 1 transition, XTRNL_RST_L needs to be forced to high to create a long reset, and low to create a short reset.

- For a **long** reset, J_POR_L deassertion happens 5 milliseconds after external POK assertion, and J_RST_L deassertion happens 512K Jbus clocks after J_POR_L deassertion.
- For a **short** reset, J_POR_L gets deasserted 32 Jbus clocks after external POK assertion, and J_RST_L deassertion happens 32 clocks after J_POR_L deassertion.

Assertion of J_POR_L and J_RST_L are both asynchronous, while deassertions are synchronous. Only the minimum amount of state is defined. The clock ratio will be reset to default. CPU starts fetching the initialization code from the Boot PROM after J_RST_L deassertion.

UPA and PCI devices and JIO's internal UPA and PCI logic get reset due to external POK, with PCI devices and JIO's internal PCI logic coming out of reset after J_RST_L deassertion. UPA logic and Devices are brought out of reset only by software.

7.1.1.1 Software-Initiated Hard Reset

By setting bit[2] in the Reset_Gen Register of JIO with JPID[2:0] = 6,7, a Sparc IIIi processor can also initiate a hard reset via JIO to all processors through J_POR_L and J_RST_L. Note that this **software initiated hard reset** can only be a long one, assertion and deassertion of J_POR_L and J_RST_L both being synchronous. J_RST_L and J_POR_L need to be asserted for 5 milliseconds for Sparc IIIi PLL to lock to new clock ratio.Deassertion of J_RST_L happens 512K Jbus clocks after deassertion of J_POR_L.

UPA and PCI devices and JIO's internal UPA and PCI logic get reset due to software initiated "hard" reset just like hardware initiated "hard" reset.

Note – Hardwire initiated Hard reset (from Power_Good) causes PLL reset in JIO and reset of Reset Source, Estar Control and Estar Init registers in JIO, while software initiated Hard Reset does not affect their state.

7.1.2 Soft Reset

When power is already on, if the "PB_RST_L" input to JIO with JPID[2:0] = 6, gets asserted due to a push-button trigger in the system, this JIO initiates a soft reset of the system by asserting J_RST_L to the processors and the other JIO.Assertion of J_RST_L is asynchronous, while deassertion is synchronous.

Note – For JIO to reliably sample PB_RST_L pin assertion and log source of reset due to this PB_RST_L assertion , system needs to ensure that the minimum assertion period for PB_RST_L pin is 10 Jbus sytem clocks.

When power is already stable and Sparc IIIi detects a transition on its J_RST_L input pins, it takes (in Sparc IIIi's terminology) a "Soft Reset". It is simply called a Soft Reset in this document. It is similar to a "Hard Reset" except that the on-chip memory controller continues to perform refresh cycles in order to preserve main memory contents, and clock ratio is unaffected. This section describes only the local POR for Sparc IIIi processors. It is labeled "local" as the rest of the system (like Memory Controller) is not reset.

There are two variants of the **Soft** reset: **long** and **short**.

To create **short** and **long** "soft" reset for the Sparc IIIis, XTRNL_RST_L pin is toggled appropriately along with PB_RST_L. During PB_RST_L 0 to 1 transition, XTRNL_RST_L needs to be forced to high to create a long reset, and low to create a short reset.

J_RST_L is kept asserted for 5 milliseconds for a long reset, and for 64 clocks for a short reset.

UPA and PCI devices and JIO's internal UPA and PCI logic get reset due to external PB_RST_L, with PCI devices and JIO's internal PCI logic coming out of reset after J_RST_L deassertion.UPA logic and Devices are brought out of reset only by software.

7.1.2.1 Software-Initiated Soft Reset

By setting bit[0] in the Reset_Gen register of JIO, a Sparc IIIi processor can also initiate a soft reset via JIO to all processors through J_RST_L.Note that this software initiated soft reset can only be a long one, assertion and deassertion of J_RST_L both being synchronous.J_RST_L needs to be asserted for 5 milliseconds for Sparc IIIi PLL to lock to new clock ratio.

UPA and PCI devices and JIO's internal UPA and PCI logic get reset due to software initiated "soft" reset just like hardware initiated "soft" reset.

Note – JIO starts calibrating its Jbus DTL I/O's after deassertion of J_RST_L and calibrates every 256 Jbus clocks. So to make sure JIO drives the read data for the first instruction fetch from the PROM correctly , system needs to make sure that the boot path is slow enough as to not return data on Jbus (through JIO) any sooner than 300 Jbus clocks after J_RST_L deassertion (for Jbus running anywhere between 120 to 200 Mhz) . This would ensure that JIO calibrated its DTL I/O's at least once before it drives Jbus. In the Enchilada,Chalupa and Taco systems which use identical boot path , the first PROM fetch takes 1955 Jbus clocks with Jbus running at 120 Mhz and 2560 Jbus clocks with Jbus running at 160 Mhz. JIO's DTL I/O's get calibrated several times within this time and drive the read data properly .

7.1.3 Externally Initiated Reset (XIR)

Sparc IIIi processor supports the SPARC V9 Externally Initiated Reset (XIR) trap. A processor takes this trap when it receives the XIR transaction over Jbus, because of signal received on input of JIO with JPID[2:0] = 6, or because Soft_XIR bit is set in Reset_Gen Register in that JIO. By definition XIR is local to the Sparc IIIi processor(s). It does not affect the rest of the system. In Sparc IIIi case in order to save some pins on both JIO and Processor module connector, XIR is transaction based on JBus.

Reset due to XIR for the Sparc IIIis does not initiate fetch of initialization code from Boot PROM, and the memory controller continues to perform refresh cycles in order to preserve main memory contents.

7.1.3.1 Soft XIR

By setting bit[1] in the Reset_Gen Register of JIO with JPID[2:0] = 6, a Sparc IIIi processor can generate an XIR to all processors. Setting that bit causes that JIO to generate an XIR transaction on JBus. Section 7.1.3.1 describes the Reset_Gen Register.The Reset Source register logs the cause of an XIR so that the XIR trap handler can easily identify the source.

Reset due to XIR for the Sparc IIIis does not initiate fetch of initialization code from Boot PROM, and the memory controller continues to perform refresh cycles in order to preserve main memory contents.

7.1.3.2 Button XIR

For bring-up purposes, the system supports a Button XIR. This reset is triggered through a push button which is connected to Southbridge and is ORed with the system_watchdog. This button is physically located on a dongle which is attached to the motherboard through a header connector.

The Button XIR feature is designed to facilitate bring-up and to provide an easy way to get the system out of software hang through an XIR instead of a general system reset. This allows to preserve most of the state of system and in particular the contents of all registers in the I/O subsystem.It can prove to be useful in identifying problems when the system hangs on I/O transfers.

The Button XIR signal is “ored” with the SouthBridge watchdog timer signal (inside the southbridge), and the result is connected to JIO’s input. When either the Button XIR or the watchdog signal is active JIO generates an XIR transaction to all processors. Bit[5] of the Reset Source register is set to one when a watchdog or Button XIR is generated. This allows the trap handler to identify the cause of the XIR.

Note – For JIO to reliably sample Button XIR assertion and log source of reset due to this XIR assertion , system needs to ensure that the minimum assertion period for XIR pin is 10 Jbus sytem clocks.

7.1.4 JIO Reset Registers

7.1.4.1 Reset_Gen Register

This register allows to reset all processors or to reset the entire system. The bits in the Reset_Gen registers are not sticky, otherwise the system will hang and will be in reset condition forever, unless the system is power cycled.

TABLE 7-1 Reset_Gen Register(JbusCSRBase + 0x41.7010)

Bit	Field	Description	Power-On Reset State	Type
0	Soft_Soft_Rst	Resets all processors and JIOs if set to 1 by asserting J_RST_L.	0	R/W (set by software, cleared by hardware)
1	Soft_XIR	Generate XIR transaction when set to 1. Neither J_POR_L nor J_RST_L are asserted.	0	R/W (set by software, cleared by hardware)
2	Soft_Pwr_On_Rst	Resets all processors and JIOs if set to 1 by asserting J_POR_L and J_RST_L.	0	R/W (set by software, cleared by hardware)
63:2	Rsvrd	Reserved	0	R

7.1.4.2 Reset_Source Register

The origin of a reset is identified by reading the contents of Reset_Source Register. When a system reset/XIR occurs a bit is set in this register so that the cause of the reset can be determined. This register must be cleared after being read to avoid ambiguity on subsequent resets. Bits are cleared by writing them to one (write one to clear).

The Reset_Source Register logs the cause of any local or external POR taken by a processor so that the POR trap handler can easily identify the source. Either of the low order three bits is set to one when the corresponding bit is set in the Reset_Gen register.

TABLE 7-2 Reset_Source Register (JbusCSRBase + 0x41.7018)

Bit	Field	Description	Power-On Reset State	Type
0	Soft_Soft_Rst	Processors and JIOs Reset through J_RST_L pin due to setting of Soft_Pwr_On_Rst bit of Reset_Gen register	0. Should preserve its value across Soft reset.	R/W1C
1	Soft_XIR	Processors received XIR through Soft_XIR bit of Reset_Gen register	0.Should preserve its value across Soft reset.	R/W1C
2	Soft_Pwr_On_Rst	Processors and JIOs Reset through J_POR_L and J_RST_L pins due to setting of Soft_Pwr_Up_Rst bit of Reset_Gen register	0. Should preserve its value across Soft reset.	R/W1C
3	Power_On (Low to High transition on Power_Good)	System reset by being powered-up. J_POR_L and J_RST_L asserted to the Sparc IIIis and the other JIO.	1.Should preserve its value across Soft reset.	R/W1C
4	Push-Button Reset (Power_Good stays high while PB_RST_L asserted)	Sparc IIIis soft reset through push-button in the system. J_RST_L asserted to the Sparc IIIis and the other JIO.	0.Should preserve its value across Soft reset.	R/W1C
5	Button XIR/ WatchDog Timer Expired	Processors receive XIR due to Button XIR/Watchdog Timer Expiration.	0.Should preserve its value across Soft reset.	R/W1C
6	Fatal Error	JIO causes a Soft reset of the system by asserting J_RST_L. Fatal error gets logged due to detection of Jbus Address Parity Error or Jbus Control Parity Error in JIO or when JIO detects fatal error from another Jbus port by sampling 3'b111 on j_pack inputs for 4 consecutive clocks.	0. Should preserve its value across Soft reset.	R/W1C
63:7	Rsvd	Reserved	0	R

Note – Programming Note: The Reset Source and Reset Gen registers are only valid for JIO with JPID[2:0] = 6, as that JIO can only initiate resets for the system.Software should not program the Reset Gen register and should not sample the Reset_Source Register in JIO with JPID[2:0] = 4.

7.1.5 UPA Reset

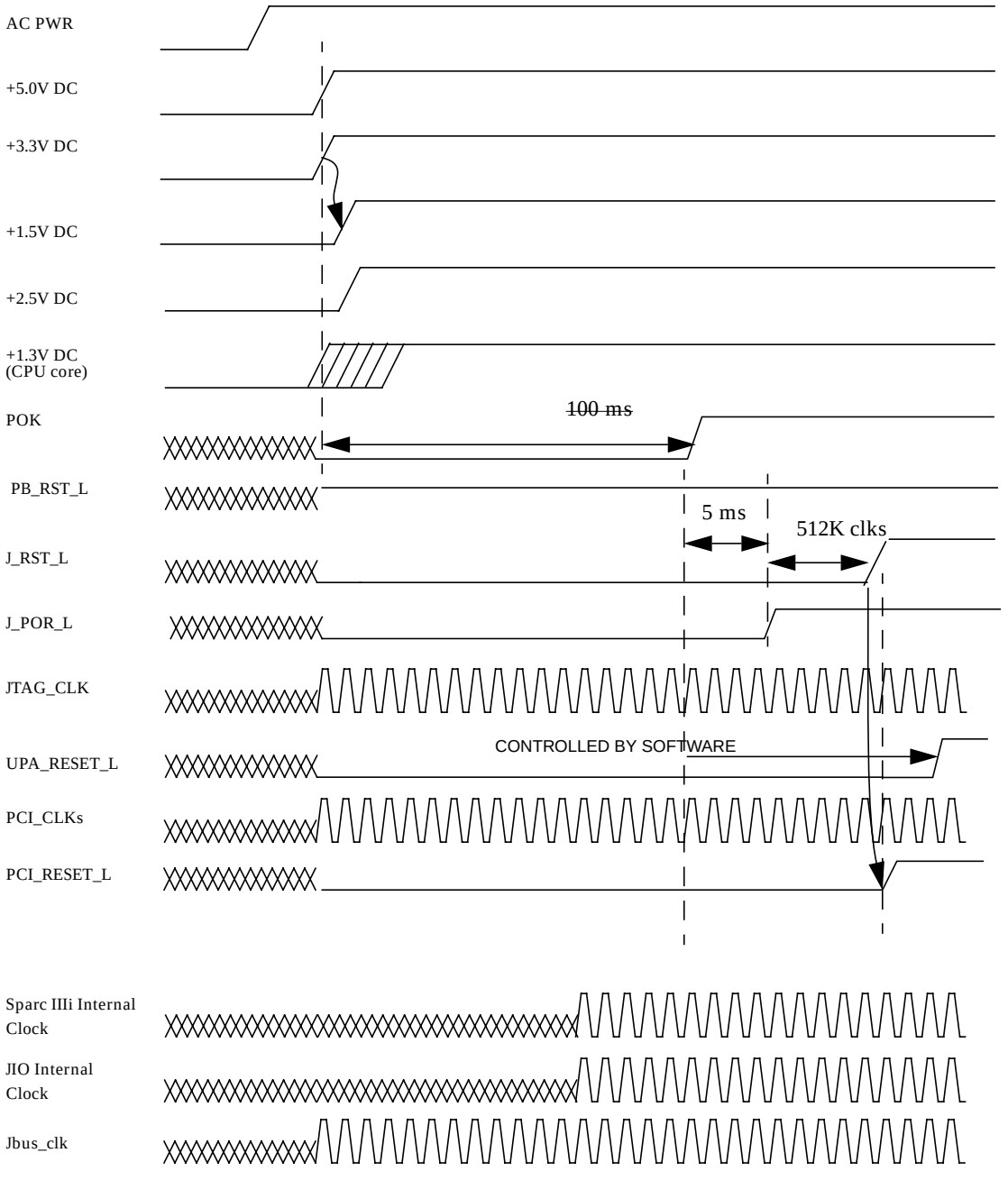
The UPA device and JIO's internal UPA logic is kept in reset by software by a bit in the UPA Reset Control Register. The reset gets deasserted when software sets that bit. Software will take care of timing between deassertion of UPA_POK and deassertion of UPA_Reset. Please refer to section 4.2.13 of the JIO PRM (UPA64S Reset Control Register) for details of UPA Reset deassertion by Software.

7.1.6 Reset Assertion During Estar Sequence

It is illegal to assert J_RST_L during Estar transition mode (J_CHNG_L) sequence, as the system behavior is undefined in such a case. J_RST_L tends to preserve the Estar state so its associated logic cannot be reset, but almost all other logic would reset on J_RST_L assertion. The system might then hang and may require a J_POR_L reset. However, some fatal errors cause J_RST_L, so J_RST_L can happen anytime. If it coincides with an Estar Transition sequence, the system may hang, and will then require a J_POR_L reset.

However it is considered legal to assert J_POR_L reset during Estar transition as it would change the state of the system just like a normal J_POR_L reset.

FIGURE 7-2 Power-On “Long” Reset Timing Diagram



APPENDIX **A**

Appendix A : PCI Behaviors

This chapter contains important information about how JIO's PCI interface behaves in the context of the PCI spec and what PCI 2.2 spec features JIO does not support/ has not implemented. The purpose is to promote in general a better overall understanding of the part in the context of system debug and software development.

A.1 Feature Compliance with PCI 2.2 Spec

JIO PCI is fully complaint with 2.2 PCI spec, but not support the following general PCI features :

- Master generated LOCK cycle
- Target or received LOCK cycle
- Master generated dual address cycle
- Master initiated fast back to back cycles
- Interrupt acknowledge cycles
- Target received special cycles
- Target received IO cycles

A.2 Support for Delayed Transactions

JIO PCI support single delayed read request, but does not support multiple delayed read request . JIO also does not support delayed write request.

A.3 M66EN Pin polarity needs to match external bus frequency

JIO PCI core runs at 132 Mhz , which is x2 or x4 of the external bus frequency (66MHZ/33MHZ).The x2 or x4 option is controlled by M66EN pin. If the M66EN pin is tied high, the x2 feature is enabled and if it is tied low, the x4 feature is enabled. The polarity of the M66EN pin has to match the external bus frequency, otherwise the internal PLL will fail to the lock the frequency.

A.4 JIO PCI arbiter supports pre-grant

To ensure the fairness and the best bus utilization, JIO PCI arbiter supports the pre-grant feature.

The pre-grant feature is that the arbiter would move the grant to the next highest priority requested master once the current master active the frame. The next highest priority master can receive the grant while the current master is still holding the bus.

Note – The next master should drive the bus **only** when the bus is IDLE and it still owns the grant. Otherwise there would be contention on the PCI bus between the current master and the next master.

A.5 JIO PCI arbiter reassigns grant to next highest priority request sooner than what the PCI spec recommends

To ensure the fairness and the best bus utilization, JIO PCI arbiter moves the grant to the next highest priority requested master if the current master did not active the bus two cycles after it receives the grant from the arbiter. The PCI spec says the current master should be able to hold on to its grant for 16 PCI clocks before it loses it to another master. But JIO takes away the grant in 2 cycles ,and hence is more stringent than the PCI spec. This was done to maximize PCI bus utilization in case of multiple masters arbitrating for PCI bus.

A.6 JIO as a PCI master backs off the bus on losing its grant after completing current data beat

JIO as a PCI bus master asserts FRAME# and claims the bus one cycle after it receives the grant from the arbiter. Then it starts to sample the grant at the beginning of each data beat transfer.Once it detect that the grant has been deasserted, it would mark the current data beat as the last data beat and back off the bus after the transfer completion of the current data beat.

This is in contrast to certain PCI devices which keep driving data beats till the end of the long burst transaction even after losing grant.

A.7 JIO does not drive PCI bus during Reset

The PCI spec says that to prevent AD, C/BE# and PAR signals from floating during reset,the central resource may drive these lines during reset but only to a logic low level.

However as the “central resource”, JIO PCI core does not drive the PCI bus during reset. Since the AD[63:32], C/BE[7:4] and PAR64 tristate signals have pullups in accordance with the PCI spec, the AD[31:0], C/BE[3:0] and PAR signals are the signals floating during reset. JIO does not drive them to logic low level as the PCI spec recommends.

A.8 For multiple data-beat 32/64 bit PIO access as master , JIO asserts IRDY# only after receiving DEVSEL#

For a 64 bit PIO transfer , JIO negotiates for 64 bit transfer with the client by asserting REQ64# with FRAME# and waits for ACK64# from the client which would be asserted by the client (if it is capable of 64 bit transfers) at the same time as DEVSEL#. Once ACK64# gets asserted by the client along with DEVSEL#, JIO asserts IRDY# . Even if ACK64# does not get asserted with DEVSEL# by the client (client not capable of 32 bit transfers), JIO asserts IRDY# after seeing DEVSEL# asserted.

To simplify the design and verification task, JIO applies the same IRDY# assertion protocol to 32 bit PIO transfers in bursts(multiple data beats), namely it asserts IRDY# only after seeing DEVSEL# asserted by the client.

A.9 JIO tri-states all the grant outputs after reset deassertion until OBP enables the individual ARB_EN bits.

The ARB_EN<7:0> of PCI Control/Status register defines the enable/disable of PCI DMA arbitration. One independent bit for each potentially supported external master device on the bus.

Each of JIO's grant output enable signals is qualified by the individual ARB_EN bits. When one individual ARB_EN bit is zero, the request from the corresponding PCI device is ignored and grant to the device is tri-stated.

The reset value for ARB<7:0> is 0x00, so after reset deassertion JIO tri-states all the grant outputs until OBP enable the individual grant signals.

The system is required to put the pull-ups on all the grant signals. These pull-ups can avoid some old PCI devices trashing the PCI bus while receiving the floating grant signals after reset.

A.10 JIO's PIO access protocols to Memory,I/O and Config Space

The following table summarizes JIO's PCI transaction protocols for PIO accesses to Memory, I/O and Config Spaces in a tabular form

TABLE 7-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Config	<= 4	32/64 bit	within 32 bit address boundary	No	1 , with 32 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	<= 4	32/64 bit	across 32 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Config	> 4 and <= 8	32/64 bit	across 64 bit address boundary	No	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 7-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Config	> 8	32/64 bit	across 64 bit address boundary	No	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	<= 4	32/64 bit	within 32 bit address boundary	No	1 single-data-beat, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	<= 4	32/64 bit	across 32 bit address boundary	No	2 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 7-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
I/O	> 4 and <= 8	32/64 bit	across 64 bit address boundary	No	2 to 3 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
I/O	> 8	32/64 bit	across 64 bit address boundary	No	3 to 4 single-data-beats, with ad[31:0] pointing to least significant valid byte enable	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	<= 4	32/64 bit	within 32 bit address boundary	No	1 , with 32 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	<= 4	32/64 bit	across 32 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 4 and <= 8	32/64 bit	within 64 bit address boundary	No	2, with 64 bit address boundary	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data

TABLE 7-3 JIO's PCI PIO Transaction protocols summary

Addr Space	Xfer Size (byte)	PCI Client	BE's from Jbus	req64_l	Data Beat(s)	cbe[7:0]	ad[63:0]
Mem	> 4 and <= 8	32 bit	across 64 bit address boundary	Yes (but no ack64)	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 4 and <= 8	64 bit	across 64 bit address boundary	Yes (gets ack64)	2, with 64 bit address boundary	[7:0]point to requested bytes	ad[63:0] has valid data
Mem	> 8	32 bit	across 64 bit address boundary	Yes (but no ack64)	4, with 64 bit address boundary if ARB_PARK = 1. or 2 PIO transactions with 2 data beats per transaction , with 64 bit address boundary if ARB_PARK = 0.	[3:0] point to requested bytes, [7:4] don't care	ad[31:0] has valid data
Mem	> 8	64 bit	across 64 bit address boundary	Yes (gets ack64)	2, with 64 bit address boundary	[7:0]point to requested bytes	ad[63:0] has valid data

APPENDIX **B**

Appendix B : UPA64S Behaviors

This chapter contains important information about how JIOJIO's UPA64S interface behaves in the context of the UPA Interconnect Architecture release 2.0. specification. The purpose is to promote in general a better overall understanding of the part in the context of system debug and software development.

B.1 Feature Compliance with UPA Interconnect Architecture Specification

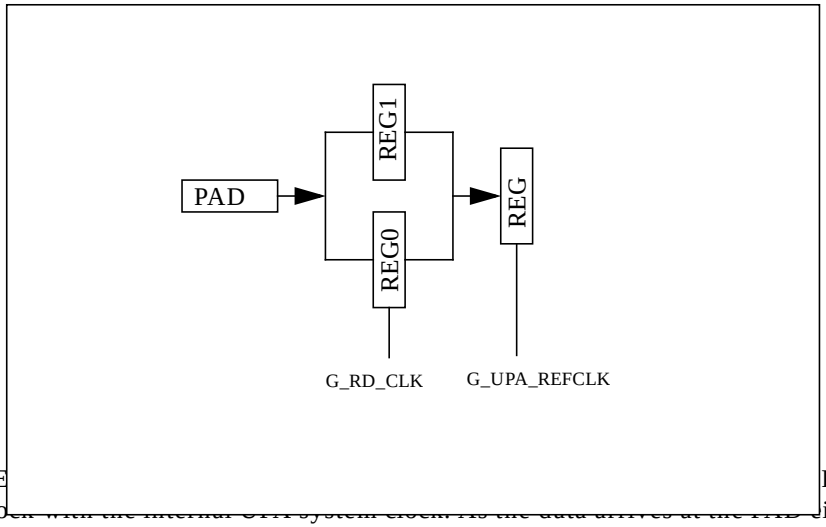
JIO UPA is fully complaint with 2.0 UPA Interconnect Architecture as it pertains to graphics slave interconnect protocol; UPA64S. There are some enhancements for performance and functionality in JIO based system as follows:

- Maximum outstanding reads are from one to four.
- Maximum outstanding writes are from one to sixteen.
- Maximum outstanding transactions are from one to sixteen.
- SPRQS, SPDQS and SQUEN controls are not required nor used.
- UPA_Slave_Int_L, UPA_IO_Speed and UPA_JTAG<4:0> port interfact signals are not used.

B.2 Support for Clock Skewing for Read Transactions

JIO supports a separate read clock for the input data and controls for compensating PC board trace delay in high speed systems.

The G_RD_CLK clock input clocks the input FIFO for JIO data and control inputs. This clock can be delayed relative to the G_UPA_REFCLK to compensate for board trace delay on the signals being read. This allows for tuning of read input valid data registration point. Below is a schematic of the input FIFO mechanism:



RE REG is clo[illegible] either input register can receive the data if it is enabled. The enable is set properly using the **UPA64s_Toggle_Read_Ptr** CSR control bit. After the data is registered in the G_RD_CLK domain it then gets registered in the REG register presenting the data to the internal UPA logic. As the data is read this UPA internal logic register will contain the first data word. The read state machine determines when it should get the data from the register and put in into the return FIFO to J-bus. This control is done using the **UPA64s_Sreply_ReadData_Latency** CSR control bit.

Combining UPA64s_Toggle_Read_Ptr and UPA64s_Sreply_ReadData_Latency CSR control bits one adjusts for the correct first data registration and proper initial data read cycles. This is done emperically in the system with the default setting at 01 respectively.

B.3 JIO Pending Transaction Min / Max

Total pending transactions were increased to provide support for current and future graphics cards. The “ONEREAD” bit mechanism was replaced with programability from one to four reads (zero is illegal). The write were limited to sixteen as a reasonable expectation of graphics card buffer sizes. Total outstanding transactions were also limited to sixteen.

Note – In development of the UPA64S graphic card queue sizing it was noted that to improve performance it would be best to have a direct control between the CPU and the graphics card which indicated when the card queues are full or near full. The latency of going through JIO, sending a write request and finding that the maximum is reached, would be avoided. This approach was not implemented.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X